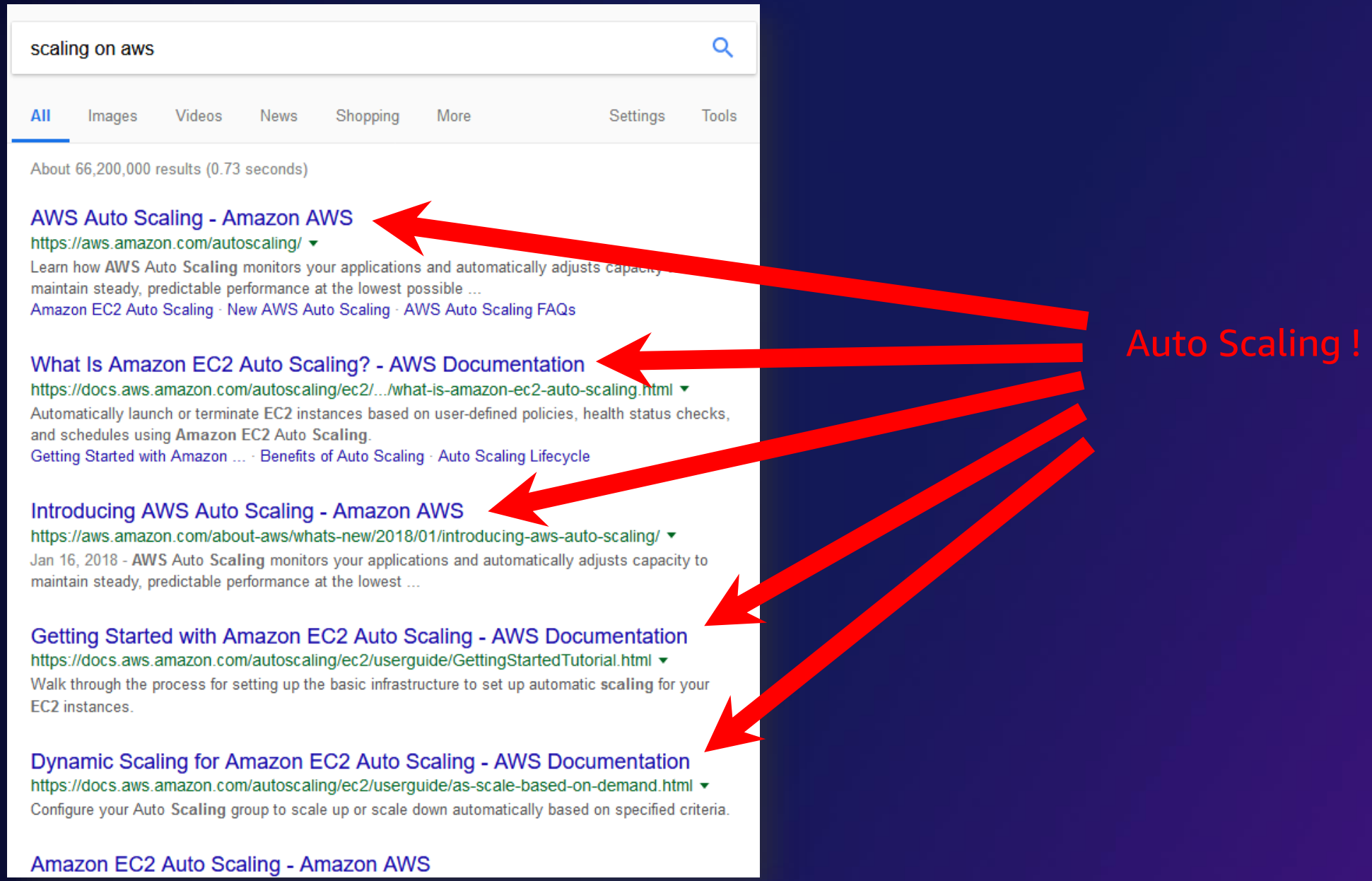

aws
SUMMIT
LONDON | APRIL 27, 2022

AR-04

# Scaling up to your first 10 million users

Dr Mike Rizzo
Senior Solutions Architect
Amazon Web Services



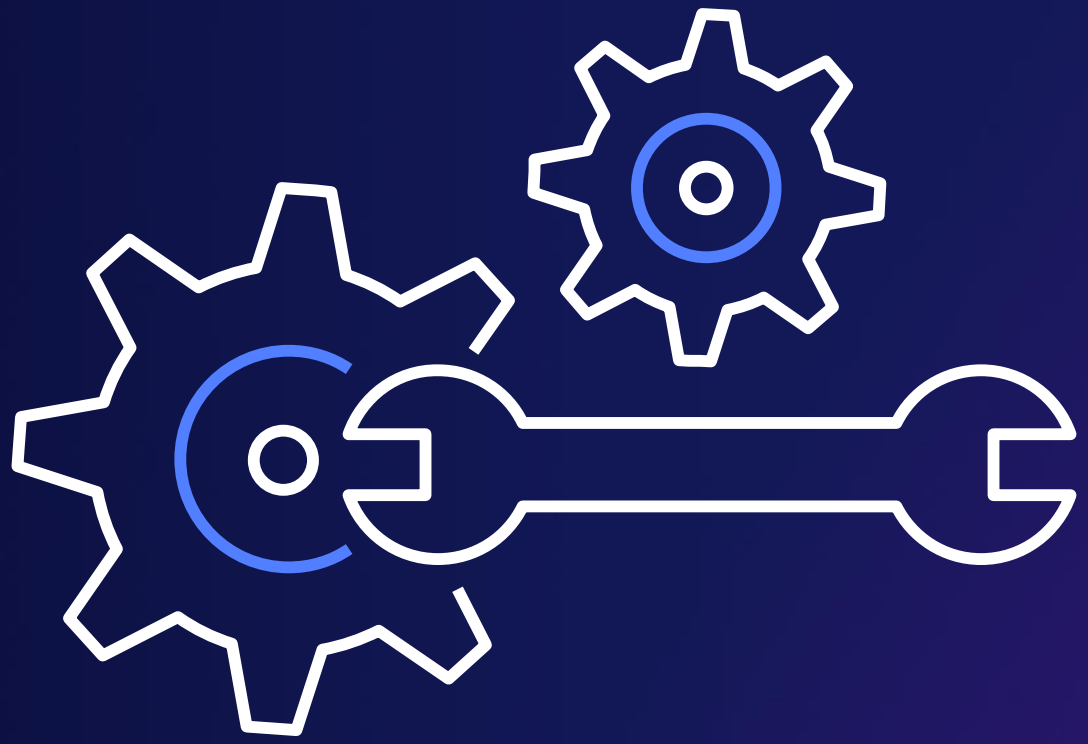

Auto Scaling !

# It’s not the single thing that fixes everything!

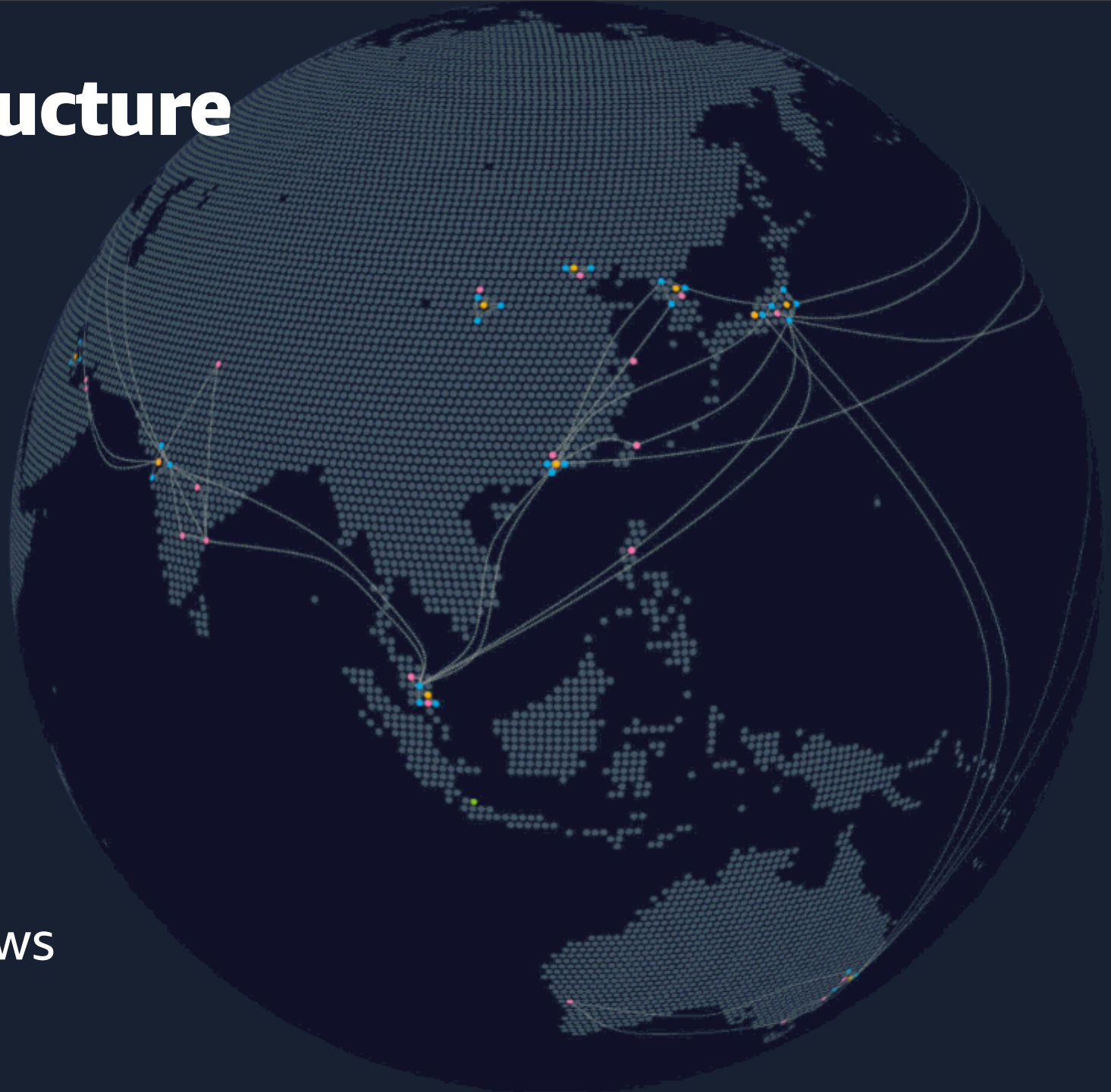
# How do I … at scale ?

- manage users
- maintain performance (incl multiple geos)
- detect and respond to incidents
- maintain business continuity
- manage security and compliance
- develop and test
- manage change
- track and manage costs
- optimize for cost
- minimize my carbon footprint

# What do we need first?



# AWS Global Infrastructure



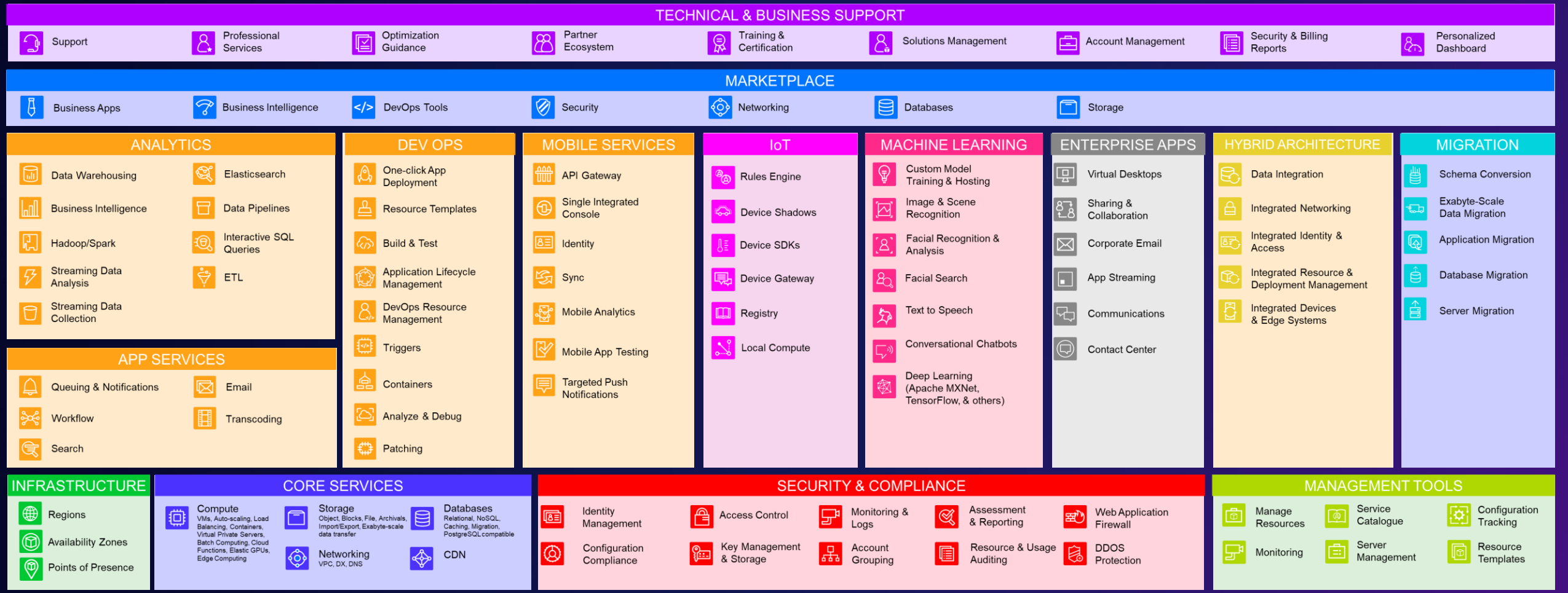
26 geographic Regions

84 Availability Zones

310+ points of presence

See more at
https://www.infrastructure.aws

# Breadth and depth of service platform



# Considerations

“Many decisions are reversible, two-way doors.”

**Jeff Bezos**
Founder and Executive Chair of Amazon

Build

Measure

Learn

# How much control do you need?

Identify and avoid undifferentiated heavy lifting

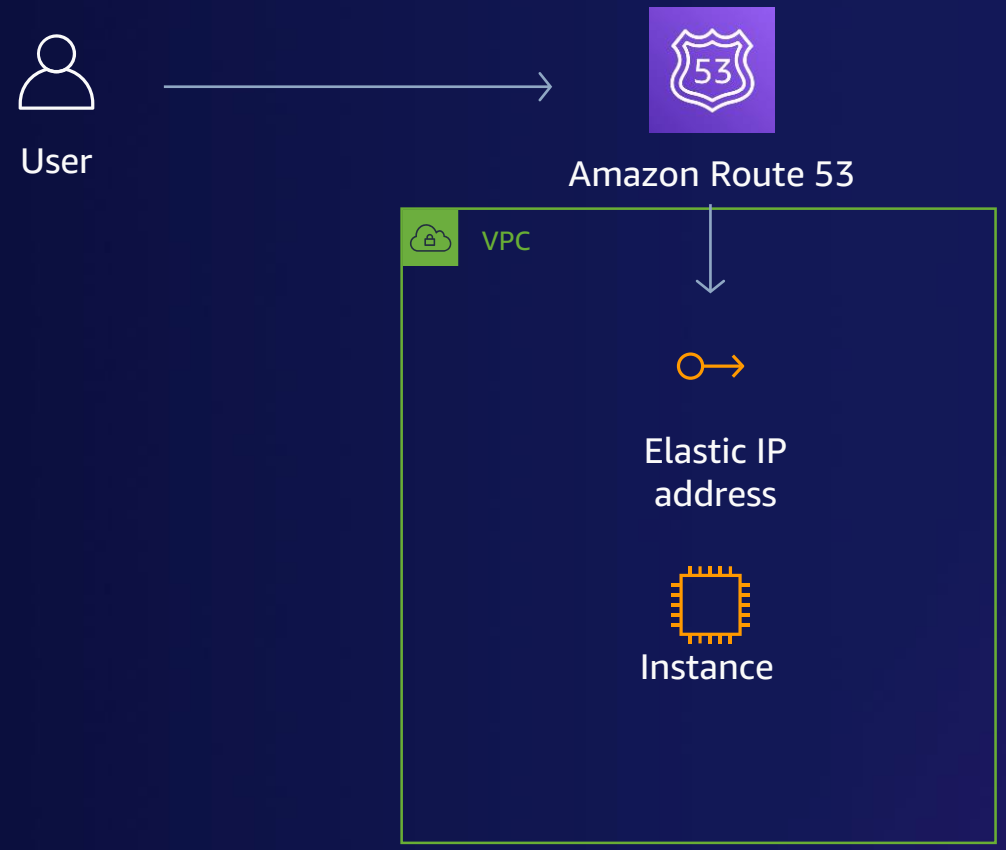
Serverless versus managed versus run it yourself

Control vs Responsibility

# So let’s start from

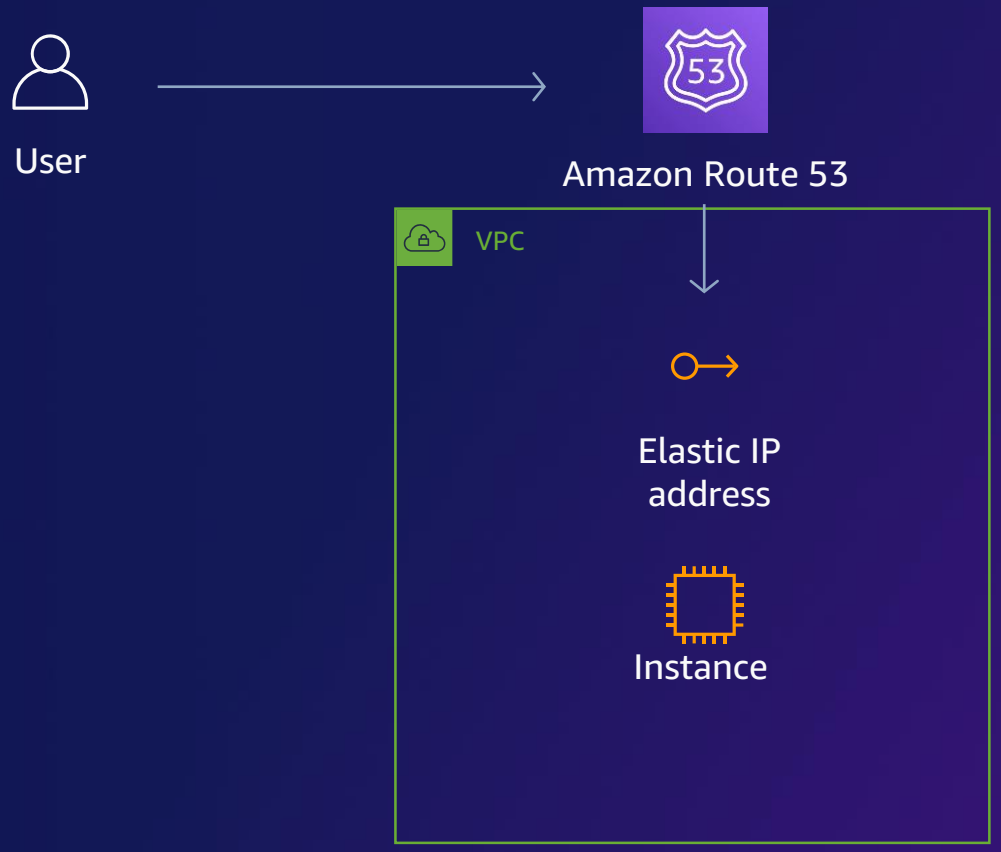
# Day 1

# 1 User : Developer / PoC



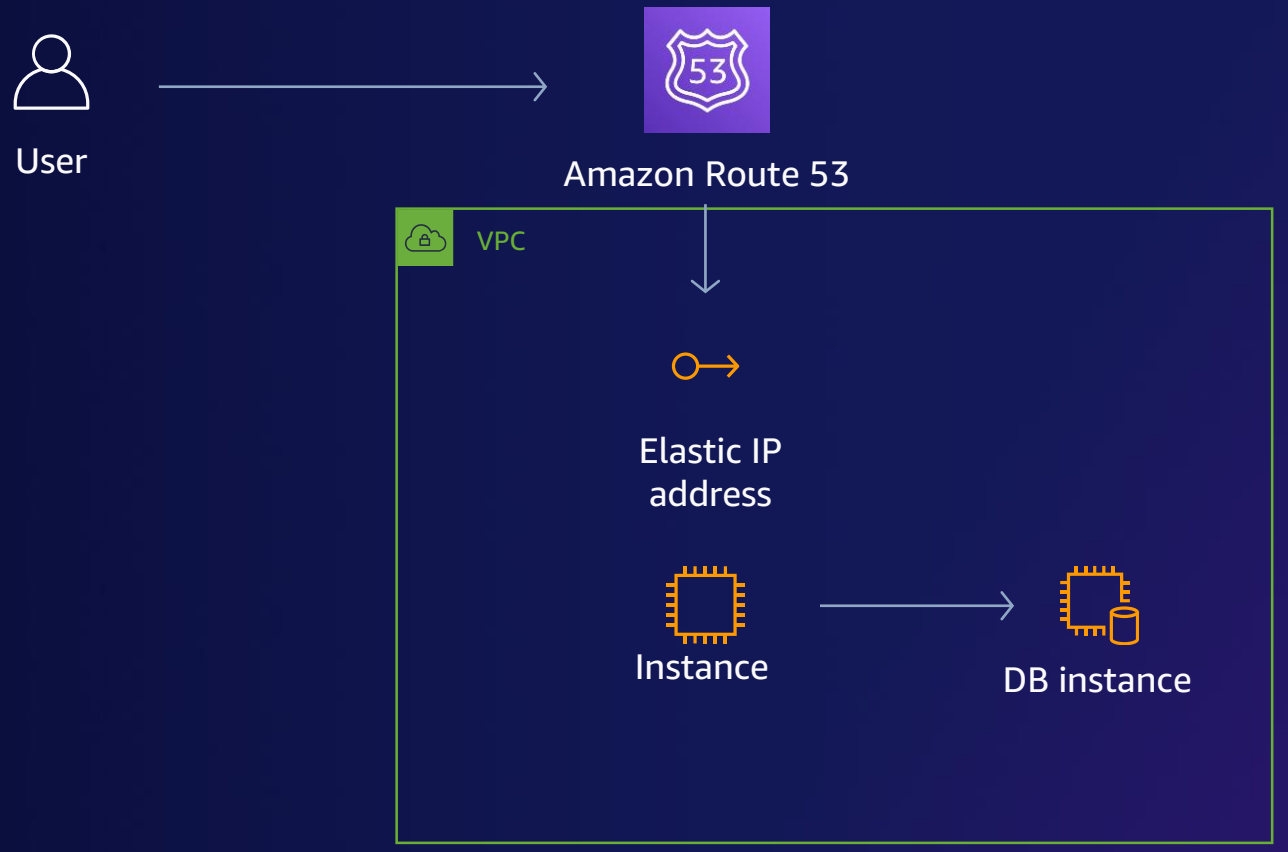
# Single instance

- No failover
- No redundancy
- Can’t scale individual components independently
- Constrained on technology choices for individual components




Too many eggs in one basket?

# Users >1 : Separate the database



# Database options

## Self-managed

Amazon EC2

## Fully managed

Amazon RDS

Amazon DynamoDB

Amazon Neptune

Amazon Aurora

Amazon Timestream

and more…

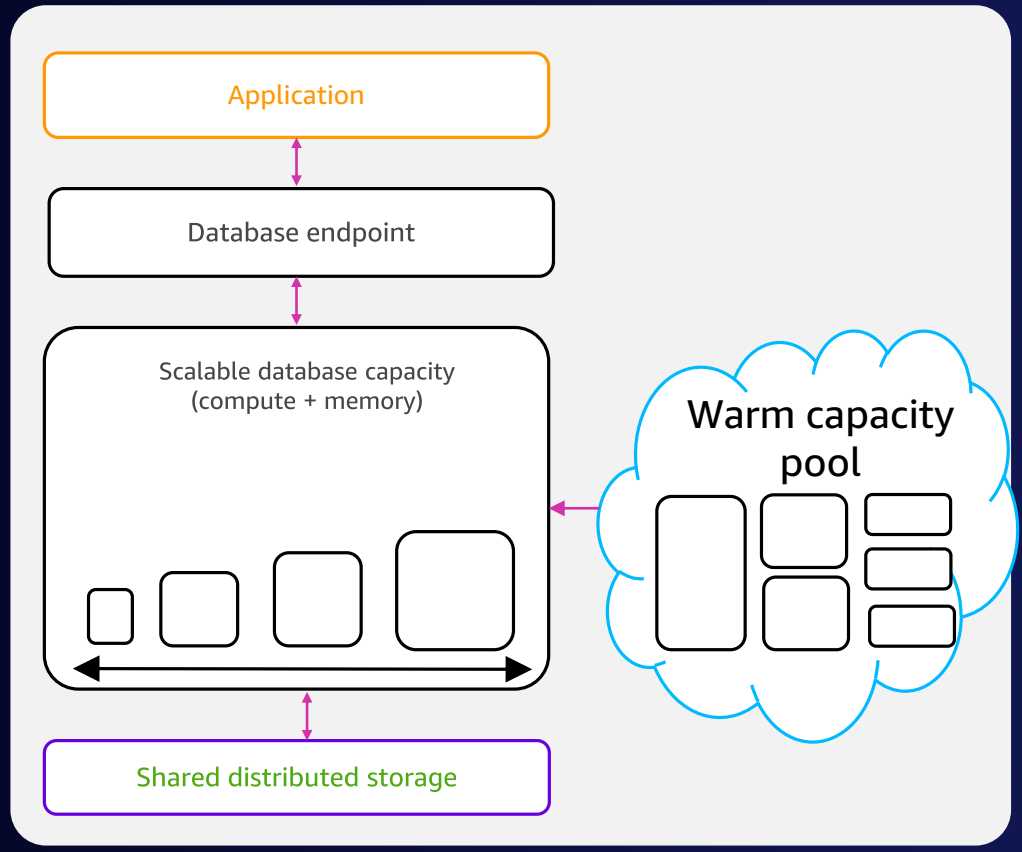
# Amazon Aurora

Amazon Aurora

- MySQL or PostgreSQL compatible
- Automatic storage scaling (up to 64 TB)
- Up to 15 read replicas
- Continuous (incremental) backups
- Six-way replication across three zones

# Aurora serverless v2 (in preview)

## On-demand, auto scaling database for applications with variable workloads




- Starts up on demand; shuts down when not in use
- Automatically scales, with no instances to manage
- Pay per second for the database capacity you use

# Relational or NoSQL?
# Or some other purpose-built DB?

It depends…

# Reasons to start with a relational DB

- Established and well-known technology
- Lots of existing patterns, code, communities, books, and tools
- You aren't going to break relational databases with your first few million users.*
- Clear patterns to scalability

*Unless you are doing something *super* peculiar with the data or you have *massive* amounts of it.

# Reasons you might need a NoSQL DB

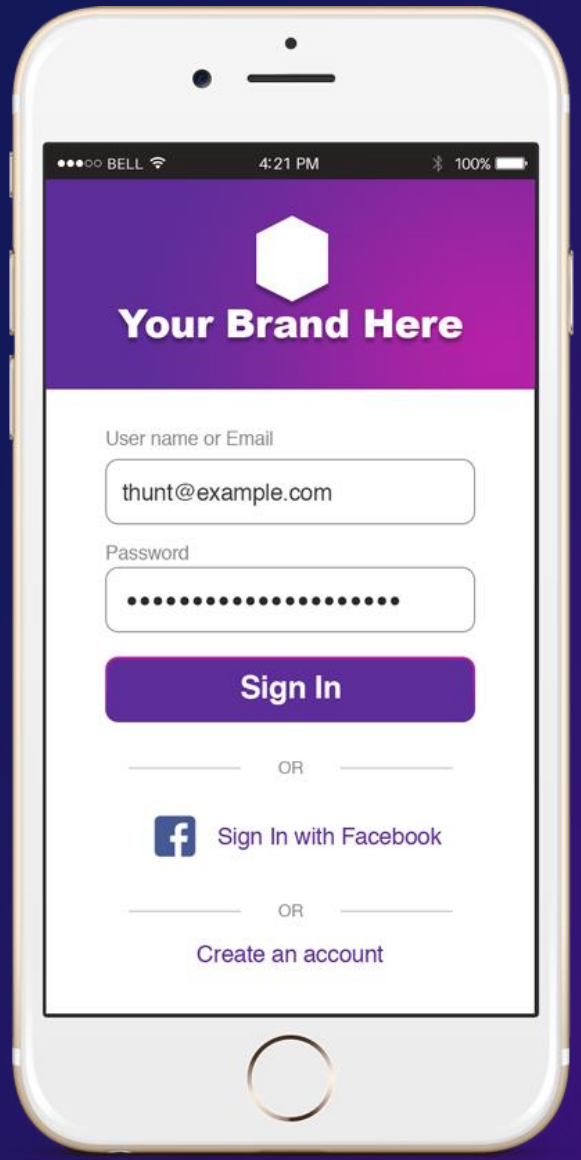
- Handle massive volumes of data (in the TB range)
- Rapid ingest of data (thousands of records/sec)
- Super low-latency applications
- Metadata-driven datasets
- Flexibility / schema-less* data constructs

- Does not mean no data modelling is required!

Need to adopt an access pattern driven approach.

# Users: >1

# Registration, sign in, and others



# Amazon Cognito overview

Web and mobile apps

*Developers focus on what is special about their app*

Amazon Cognito

*Amazon Cognito handles authorization and identity*

Managed user directory

Hosted user interface

Federation

Google SAML

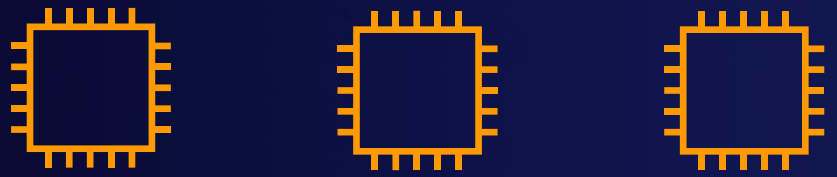
AWS credentials

Standard tokens

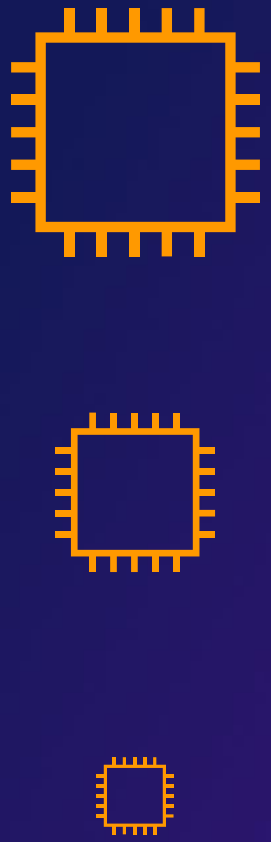
# 1 < Users < 1000

# Users > 1000 : Scaling options

Horizontally



Vertically

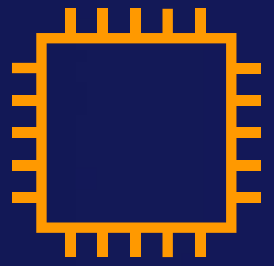


# “We’re gonna need a bigger box”

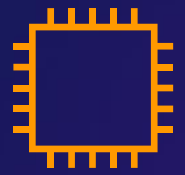
Simplest approach: vertical scaling

- Instance types:
  - General purpose
  - Memory optimized
  - Compute optimized
  - Storage optimized
  - Accelerated compute
- Easy to change instance sizes

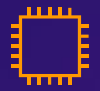
- Disk types
  - Magnetic vs SSD
  - Provisioned IOPS



c5.9xlarge



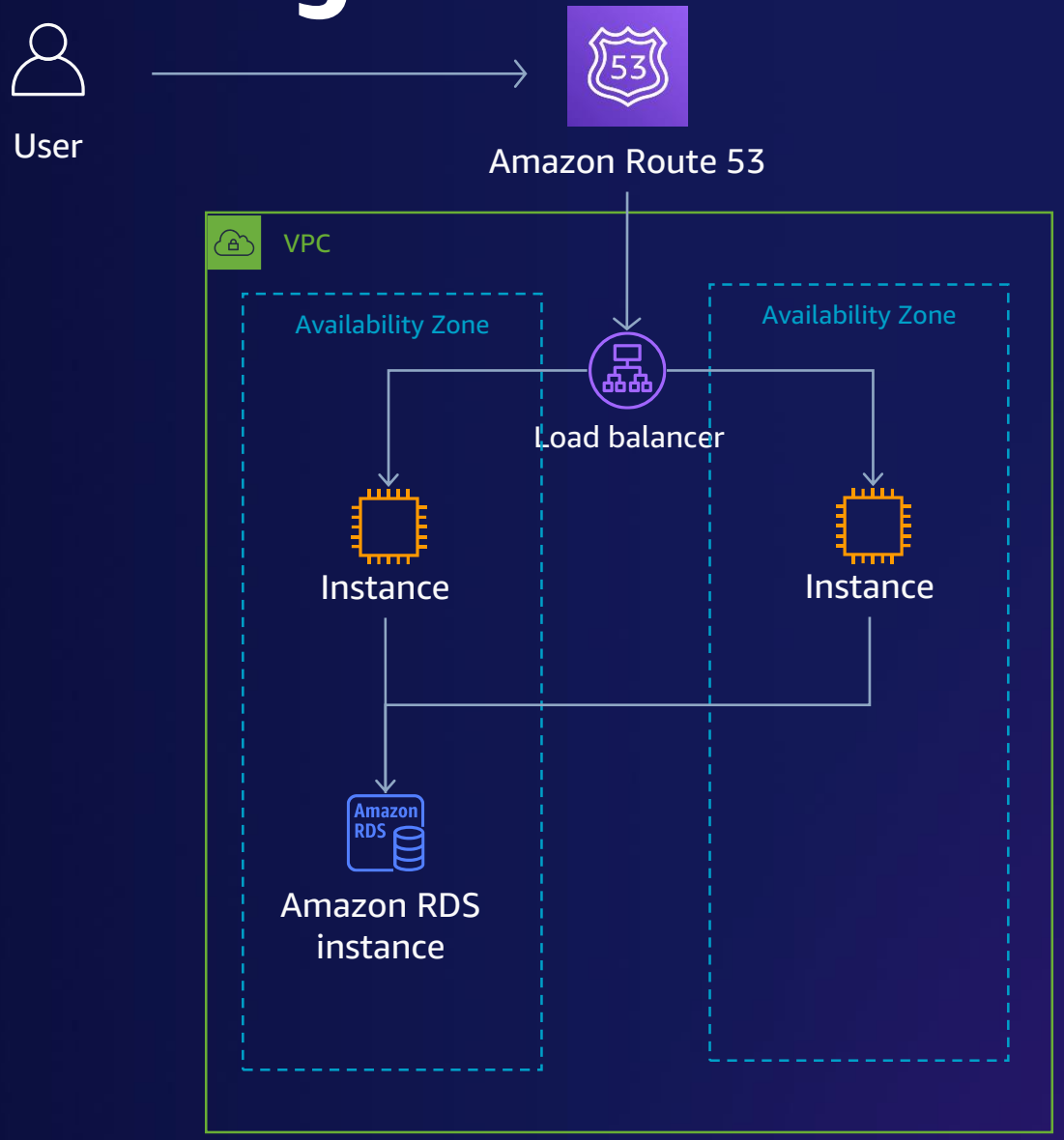
m5.2xlarge



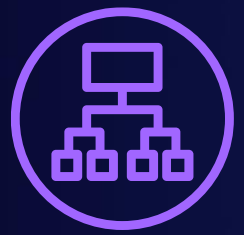
t3.nano

Will eventually hit a maximum limit!

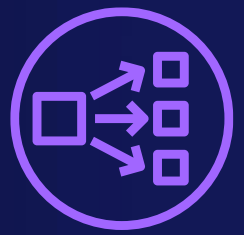
# Horizontal scaling



# Sharing the load



Application Load Balancer



Network Load Balancer

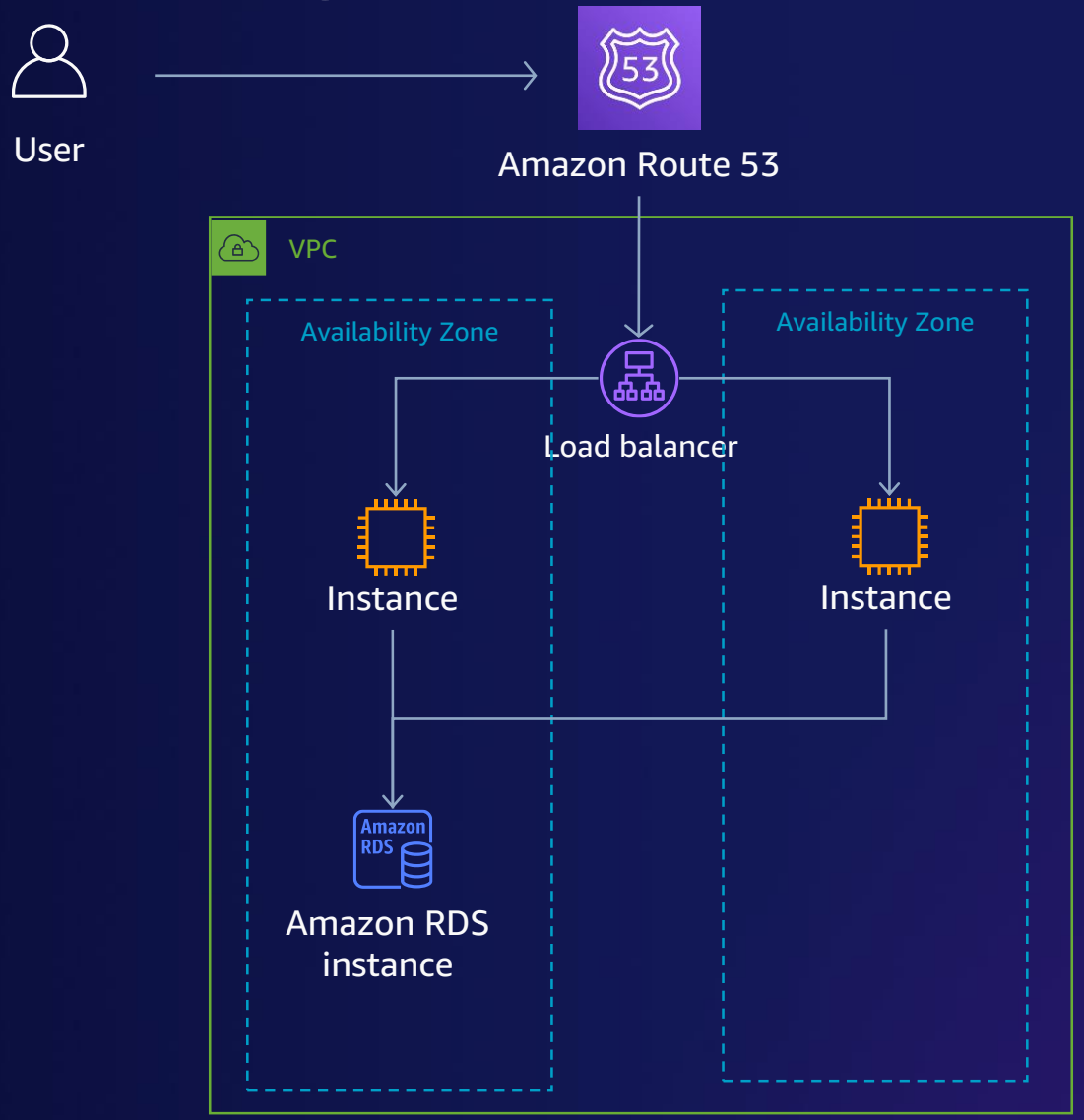
Classic Load Balancer

# Application Load Balancer

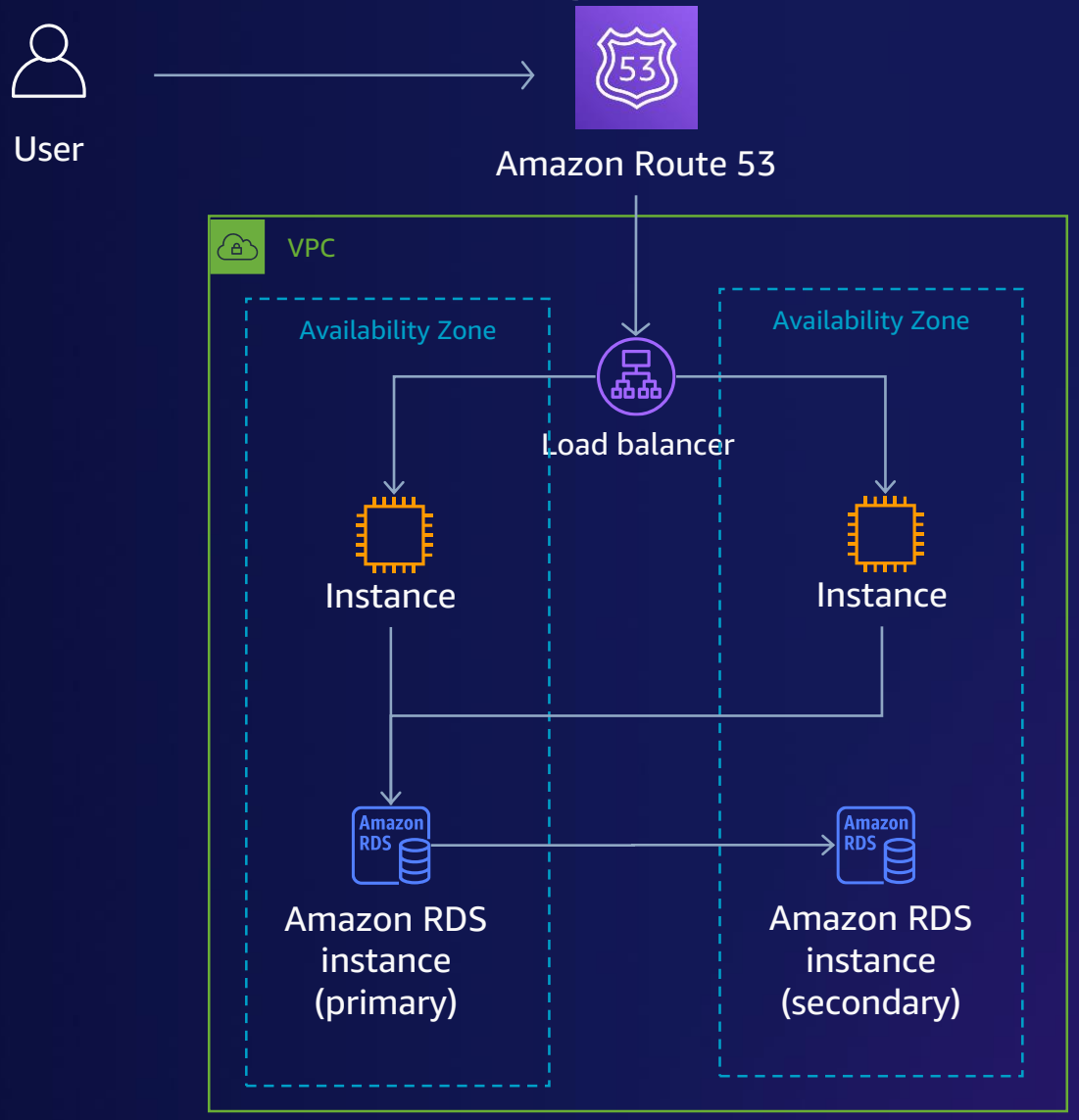
Application Load Balancer

- Highly available
- Health checks
- Session stickiness
- Monitoring/logging
- Content-based routing
- Container-based apps
- WebSockets
- HTTP/2

# Horizontal scaling

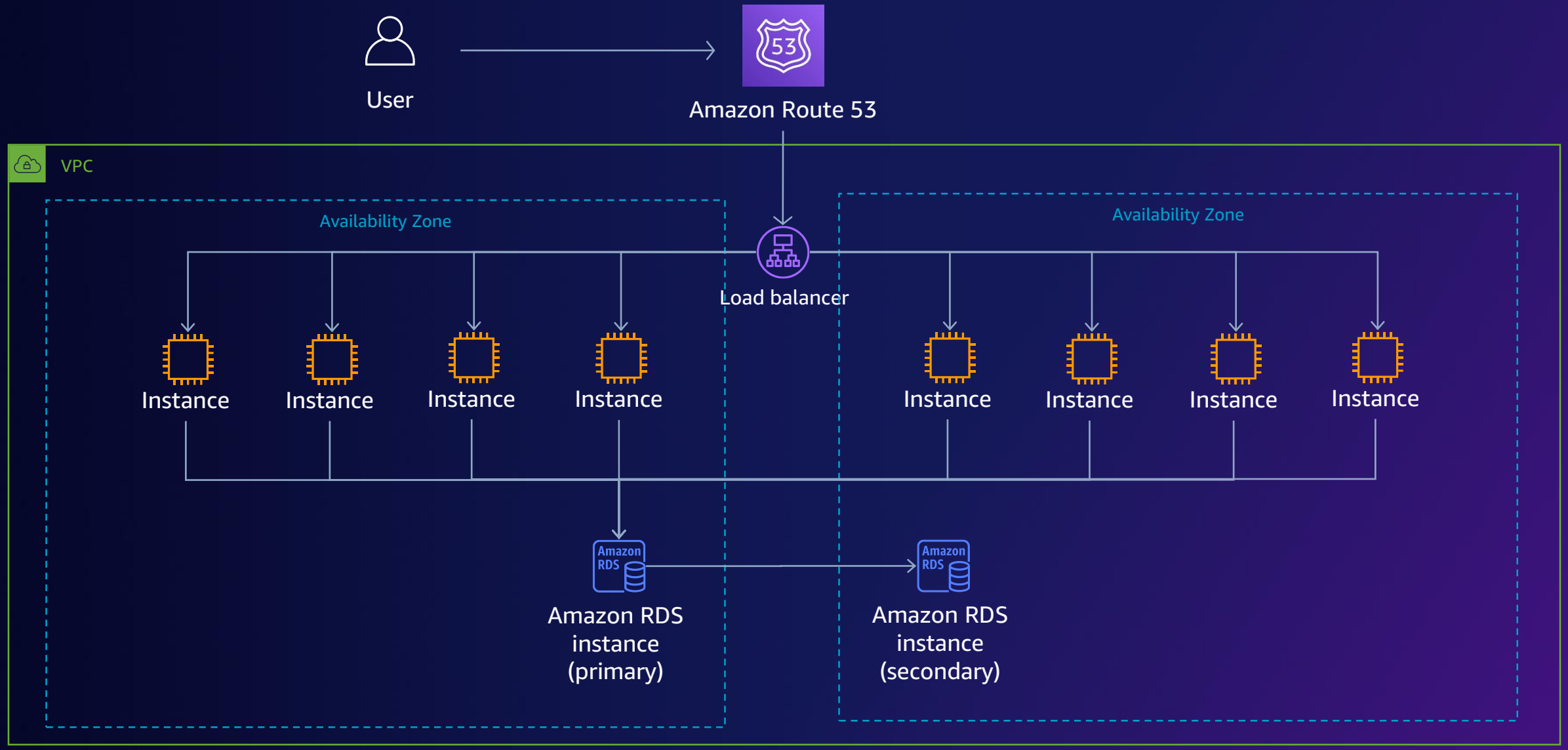


# Improve DB availability

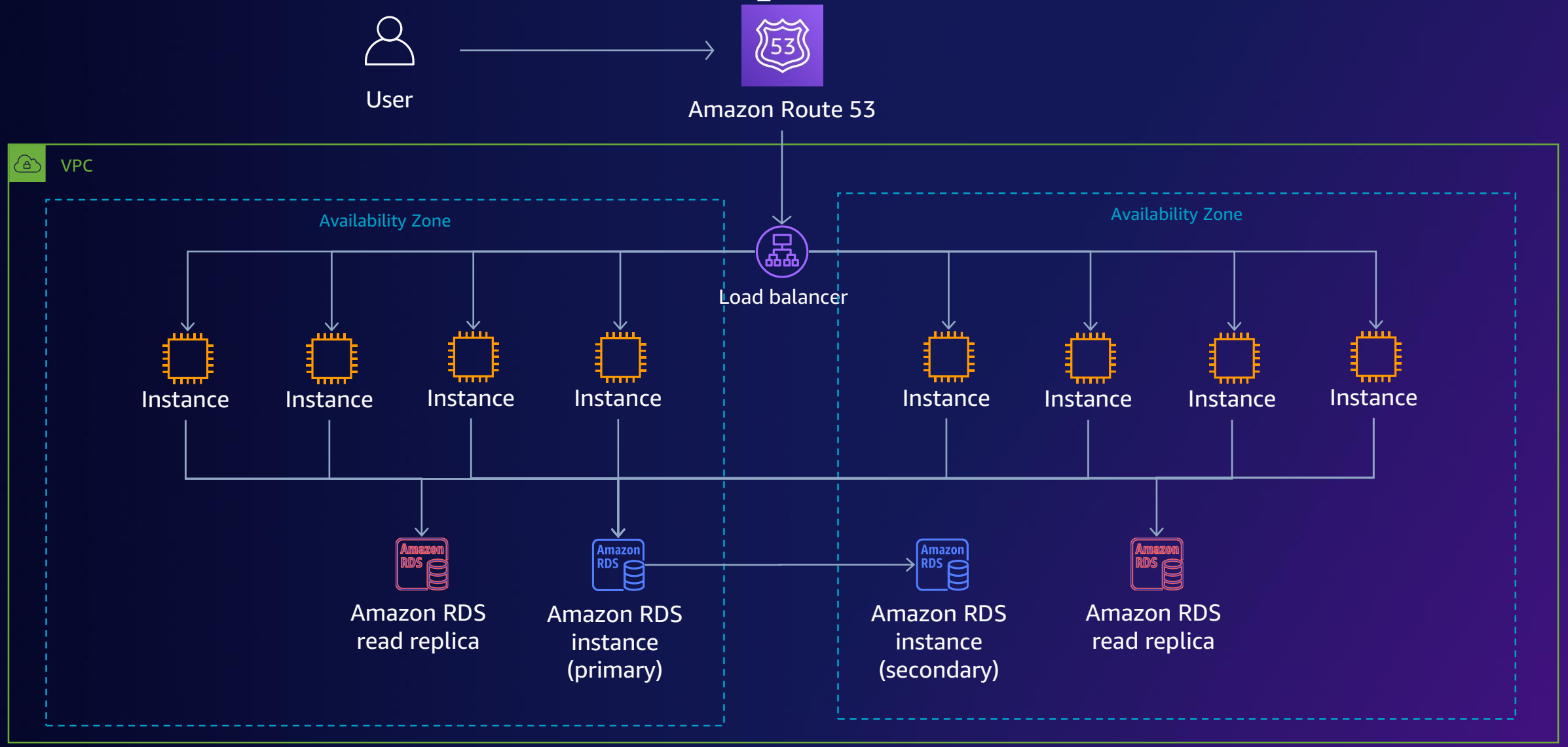


# Users >10,000

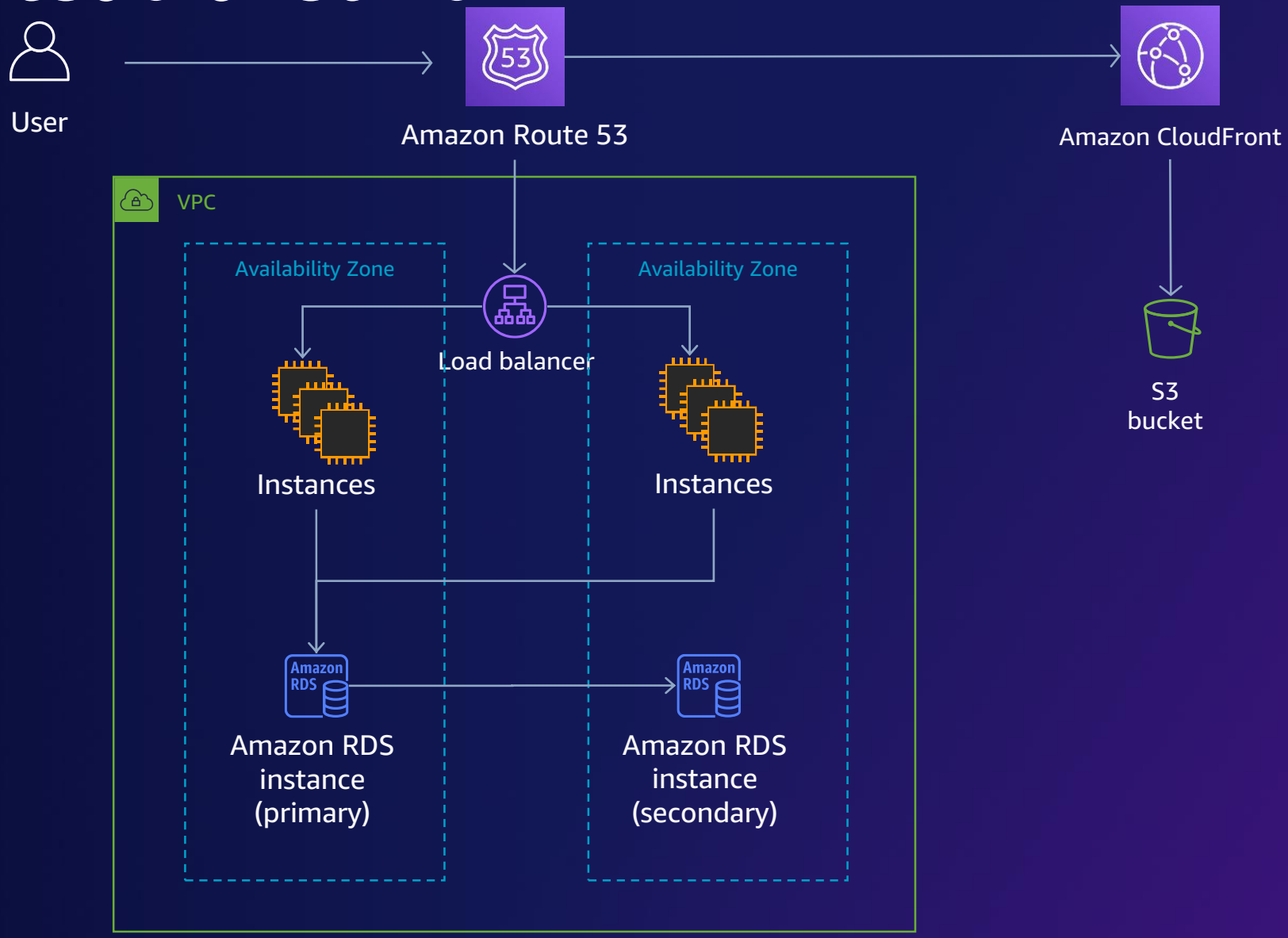
# Users >10,000



# Users >10,000 : Read replicas



# Shift some load around



# Amazon S3

Amazon S3

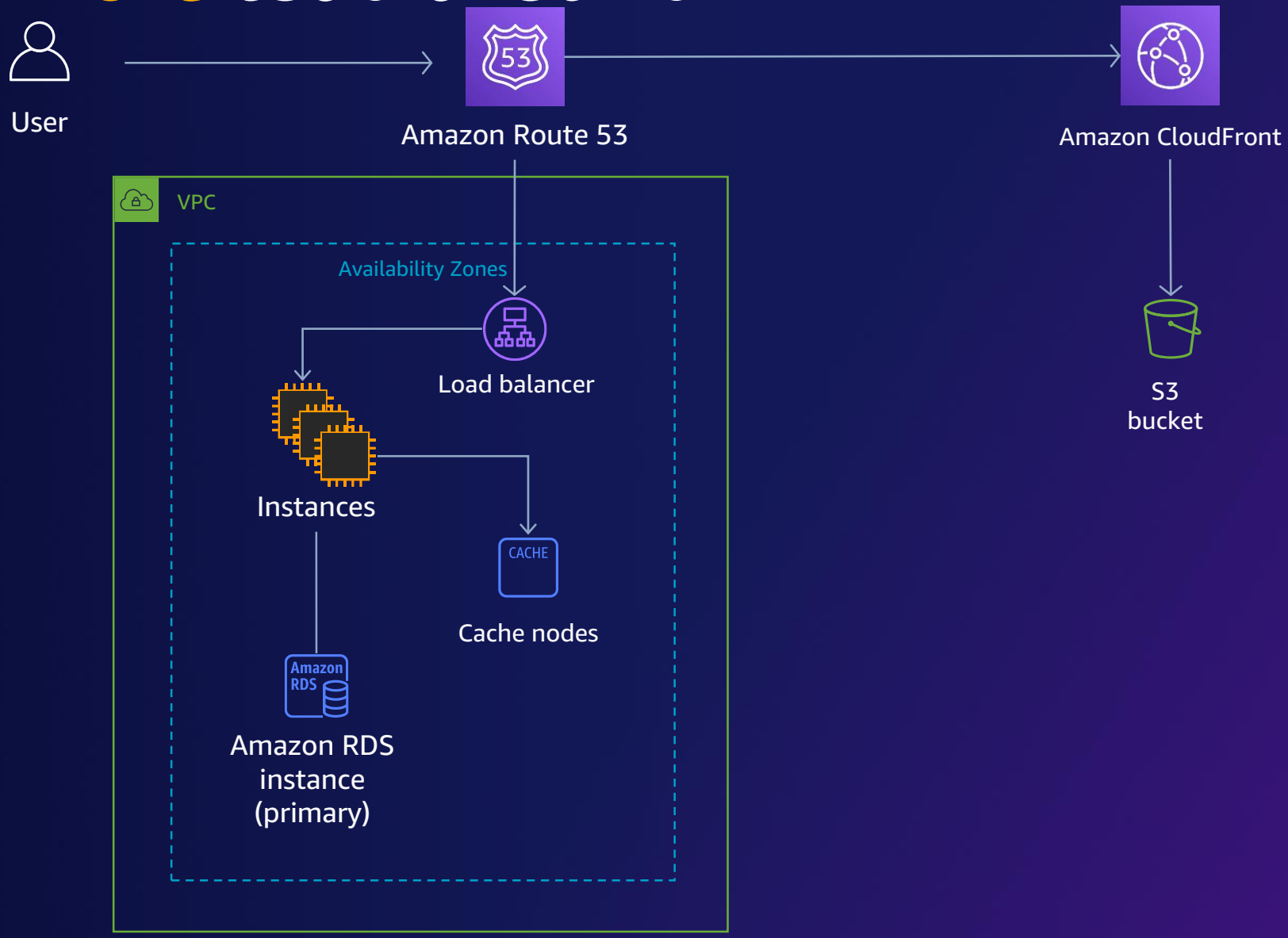
- Object-based storage
- Highly durable
- Great for static assets
- “Infinitely scalable”
- Objects up to 5 TB in size
- Encryption at rest and in transit

# Amazon CloudFront

Amazon CloudFront

- Cache content for faster delivery
- Lower load upon origin
- Dynamic and static content
- Streaming video
- Custom SSL certificates
- Short time to live (TTL) (as little as 0 seconds)
- Optimized for AWS

# Shift some *more* load around

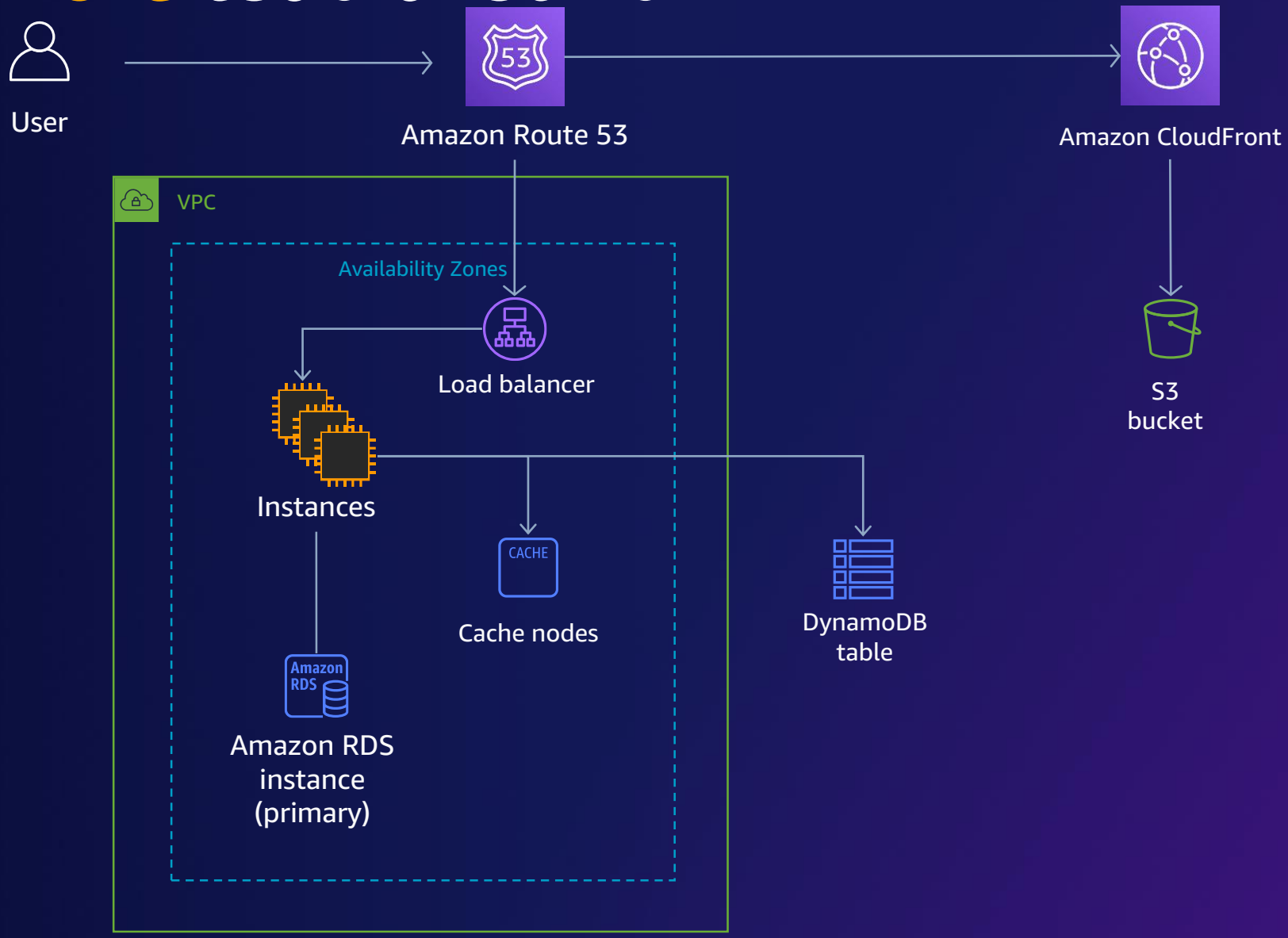


# Amazon ElastiCache

Amazon ElastiCache

- Managed Memcached or Redis
- Scale from one to many nodes
- Self-healing
- Microsecond latency
- Multi-AZ deployments

# Shift *even more* load around



# Amazon DynamoDB

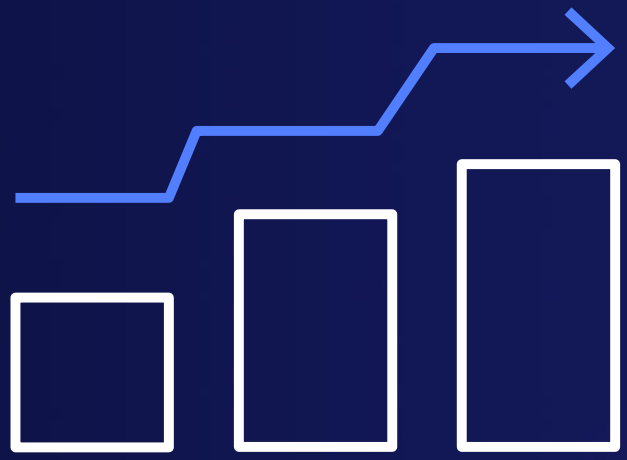
Amazon DynamoDB

- Managed NoSQL database
- Provisioned & on-demand pricing options
- Fast, predictable performance
- Fully distributed, fault tolerant
- Streams and triggers
- Global (multi-region) tables

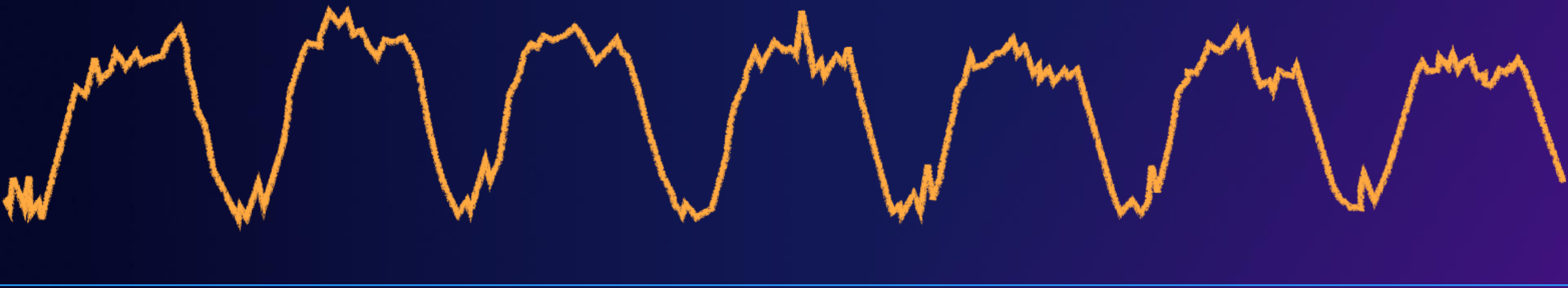
AWS Database Migration Service (AWS DMS)

Now that our web tier is much more lightweight, we can revisit the beginning of our talk…

# Auto Scaling

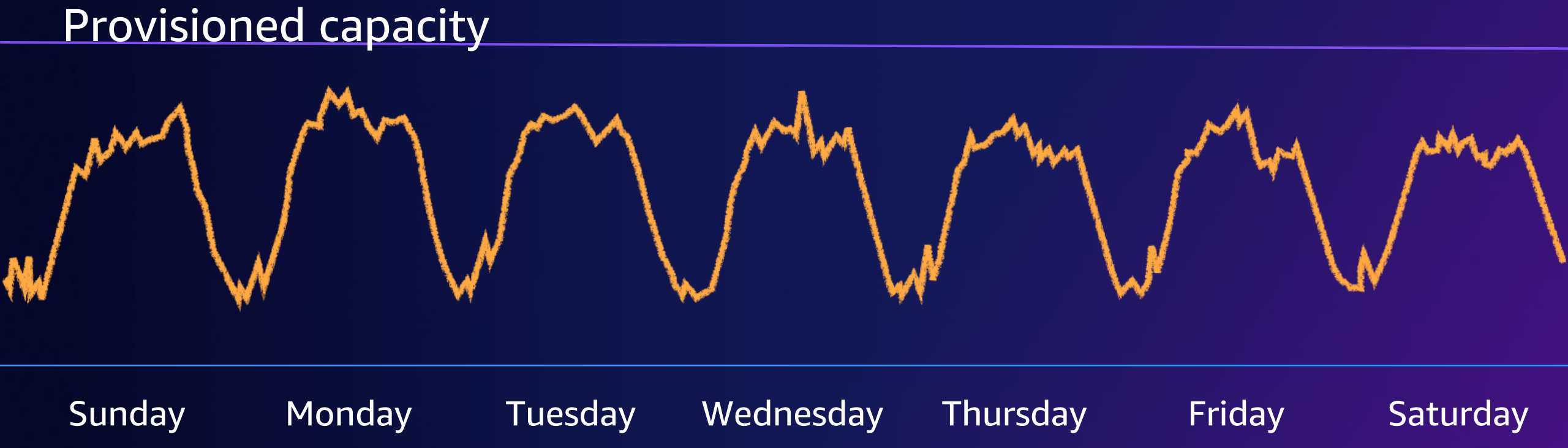


# Typical weekly traffic to Amazon.com

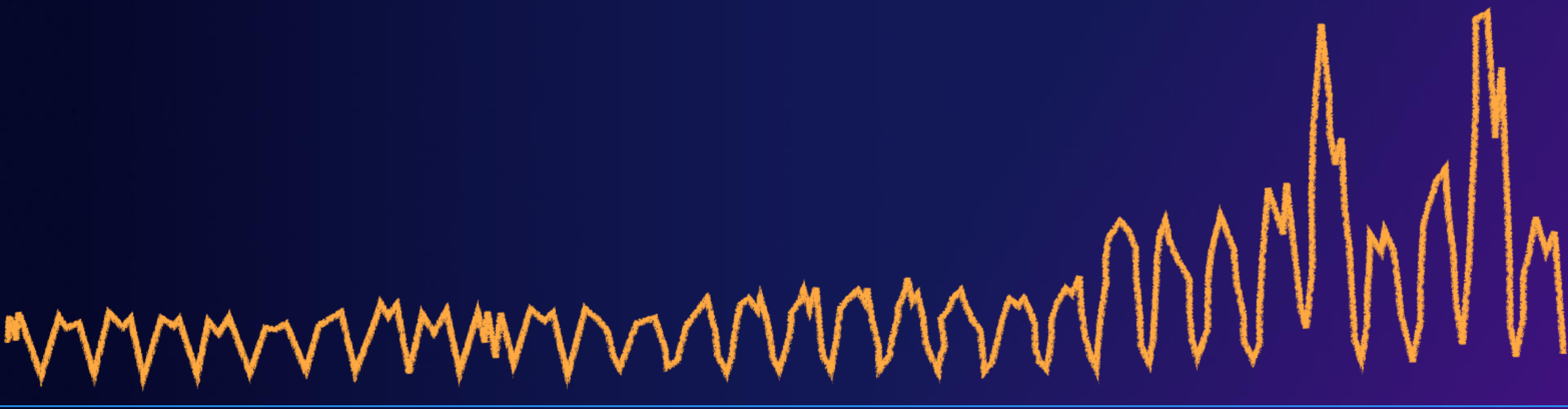


Sunday Monday Tuesday Wednesday Thursday Friday Saturday

# Typical weekly traffic to Amazon.com

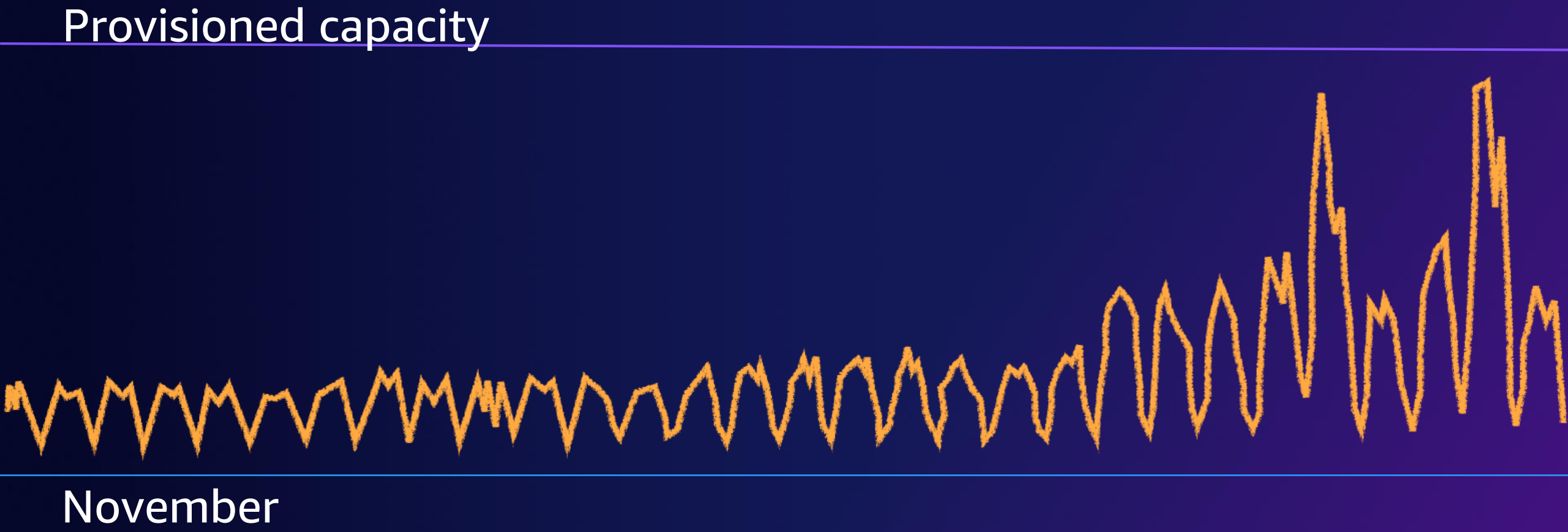


# November traffic to Amazon.com

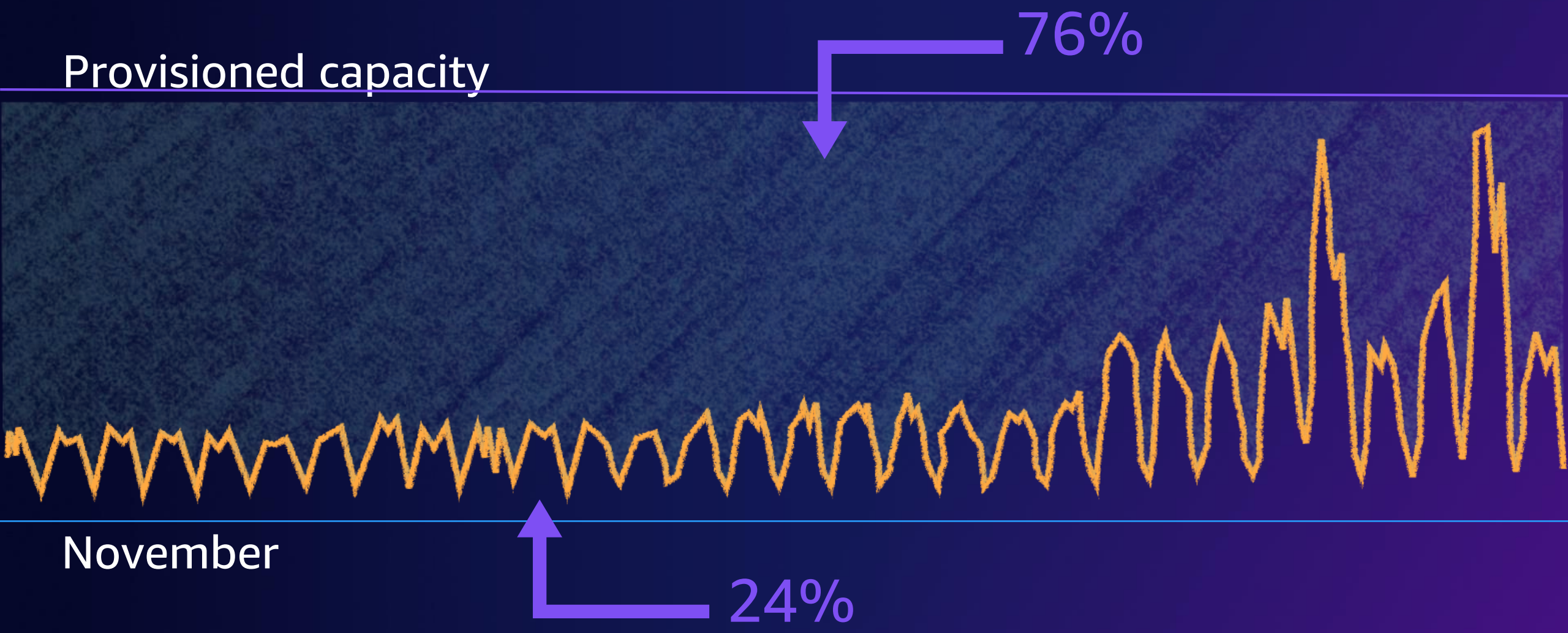


November

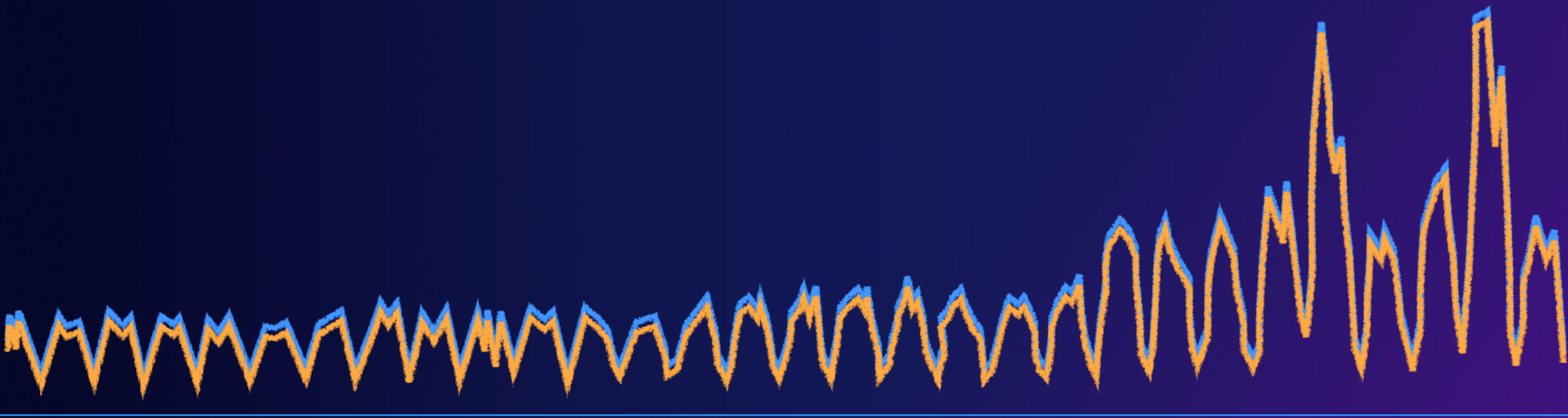
# November traffic to Amazon.com



# November traffic to Amazon.com

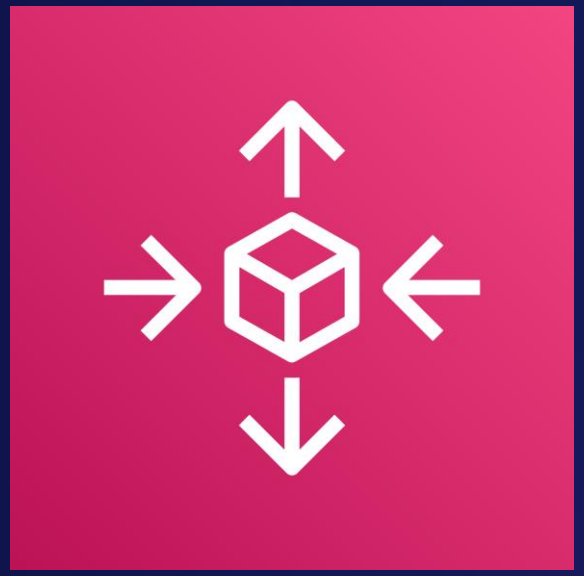


# November traffic to Amazon.com



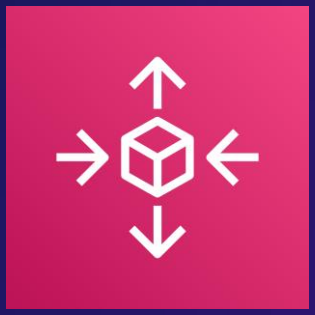
November

# Auto Scaling lets you do this

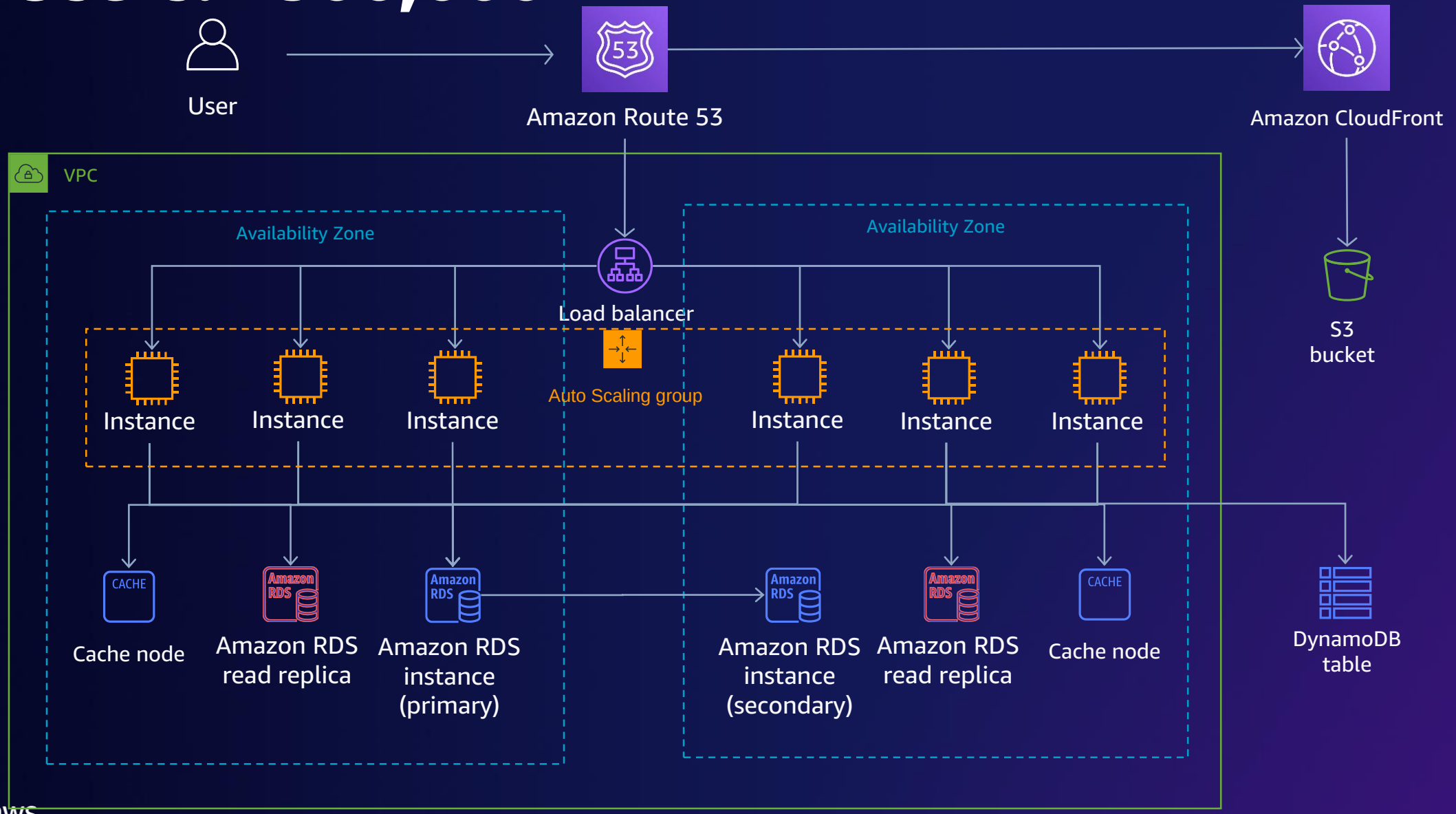


# Auto Scaling

- Automatic resizing of compute clusters
- Across multiple AZs
- Minimum/maximum pool sizes
- Amazon CloudWatch metrics drive scaling
- Replace unhealthy EC2 instances

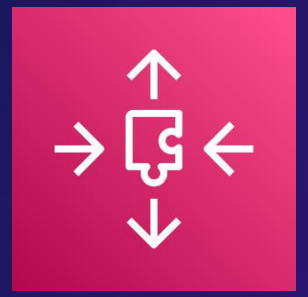


# Users: >500,000



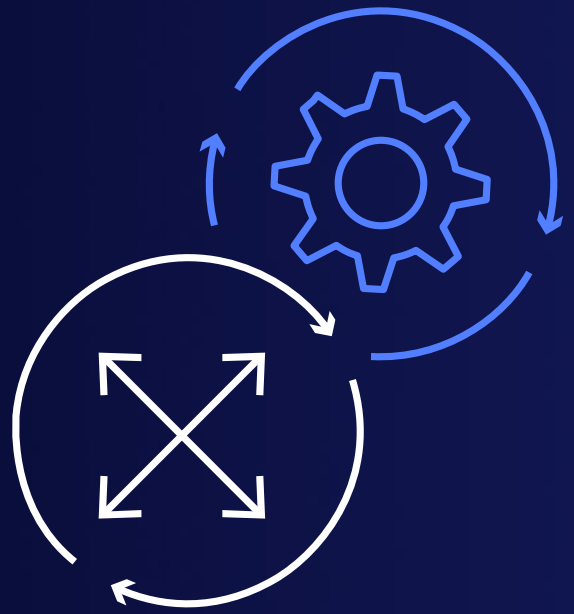
# Autoscaling is not just for EC2!

- AWS Application auto scaling for :
  - Provisioned capacity in Amazon DynamoDB
  - Amazon Aurora replicas
  - Container services running in Amazon Elastic Container Service (ECS)
  - Elasticache (Redis) replication groups
  - Numerous other AWS services
  - Custom application resources



AWS Application Auto Scaling

# Use automation!



# Automate operational tasks

AWS Systems Manager

In the cloud and on premises

Managed remote access (no bastions)

Automate common tasks

Compliance management and reporting

Basic and advanced parameter store

Incident Management

# Automate infrastructure

AWS Cloud Development Kit (AWS CDK)

AWS Tools and SDKs

AWS Command Line Interface (AWS CLI)

AWS Cloud Control API

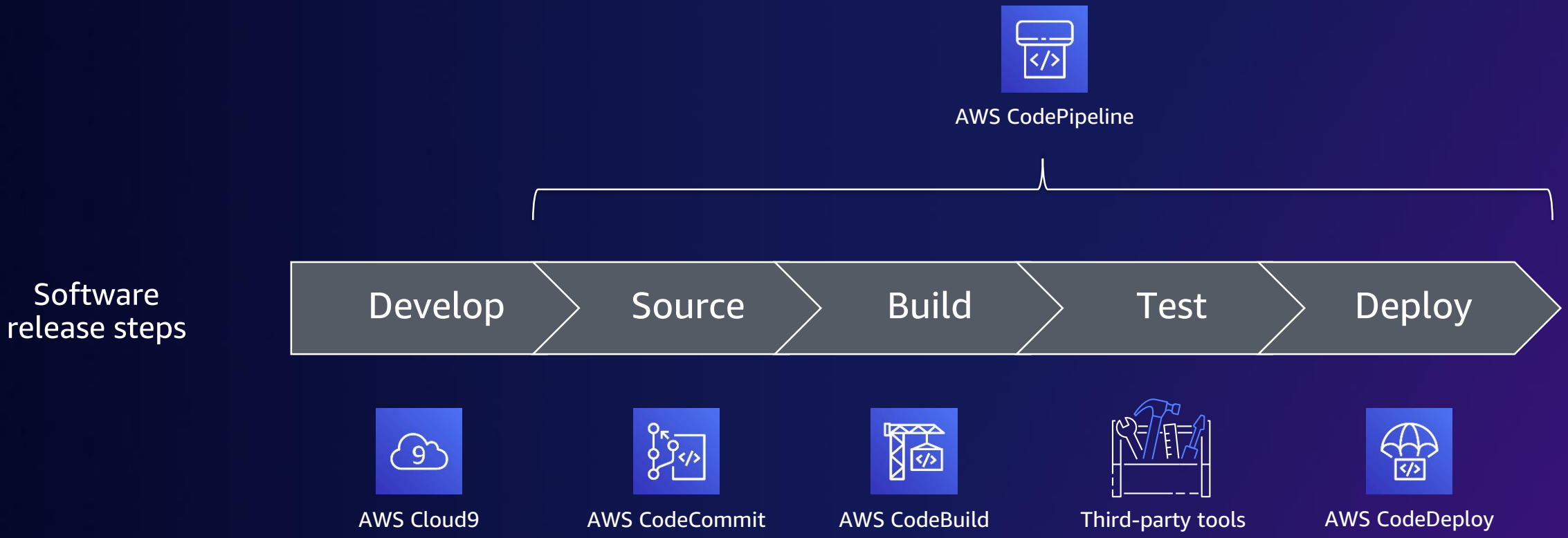
AWS CloudFormation

AWS Identity and Access Management (IAM)

AWS Service APIs

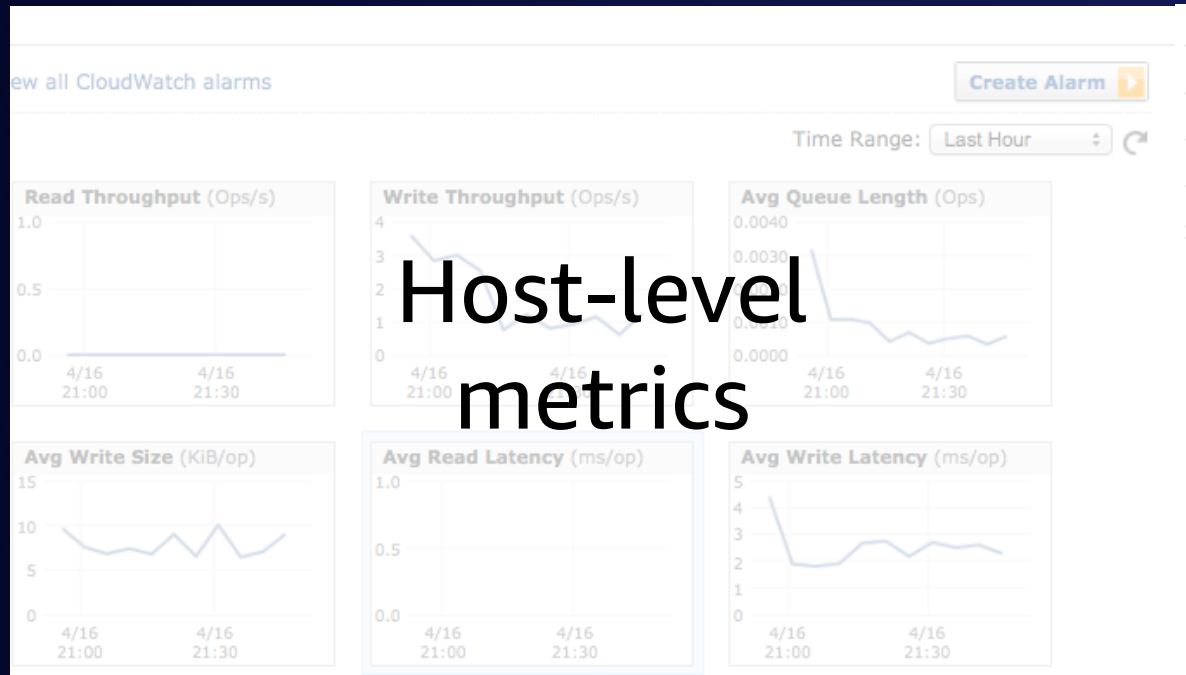
AWS CloudTrail

# Automate deployment pipeline

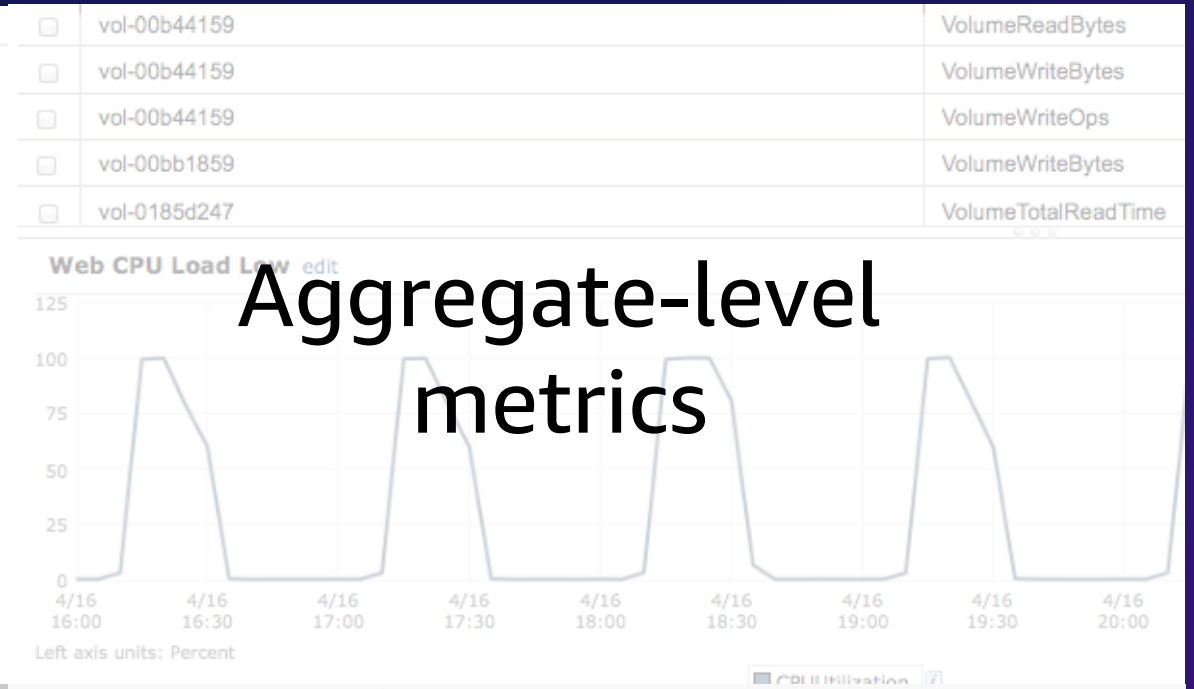


# How do I know things are going well?

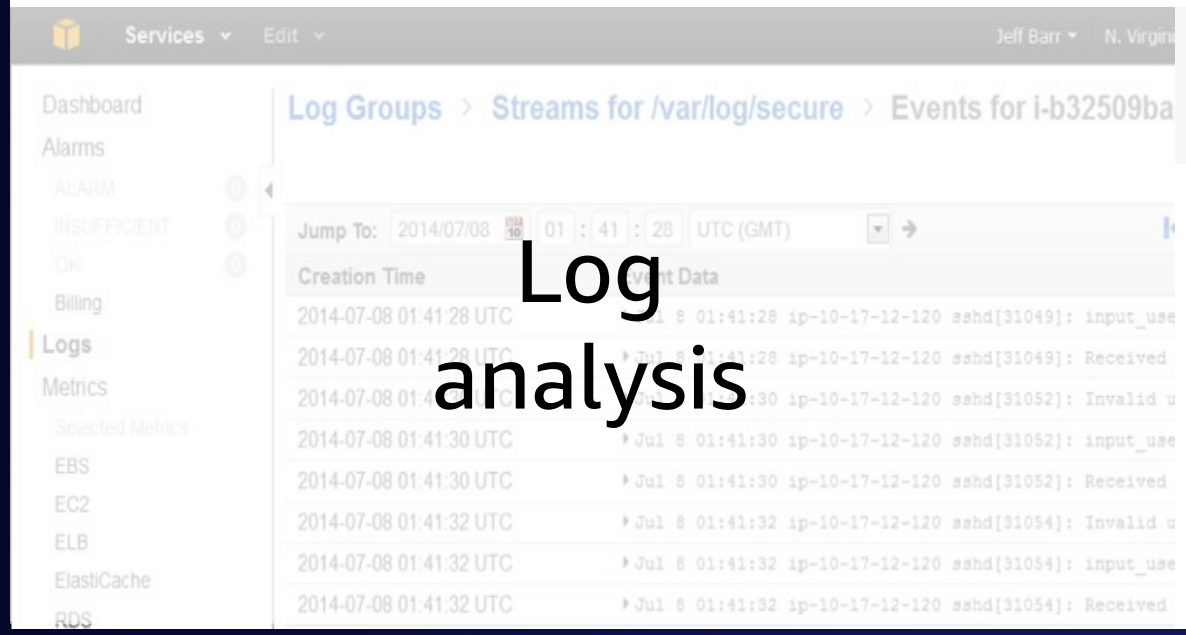
- Monitoring, metrics, and logging
  - If you can’t build it internally, outsource it (third-party SaaS)
- What are customers saying?
- Try to squeeze as much performance as possible out of each service/component



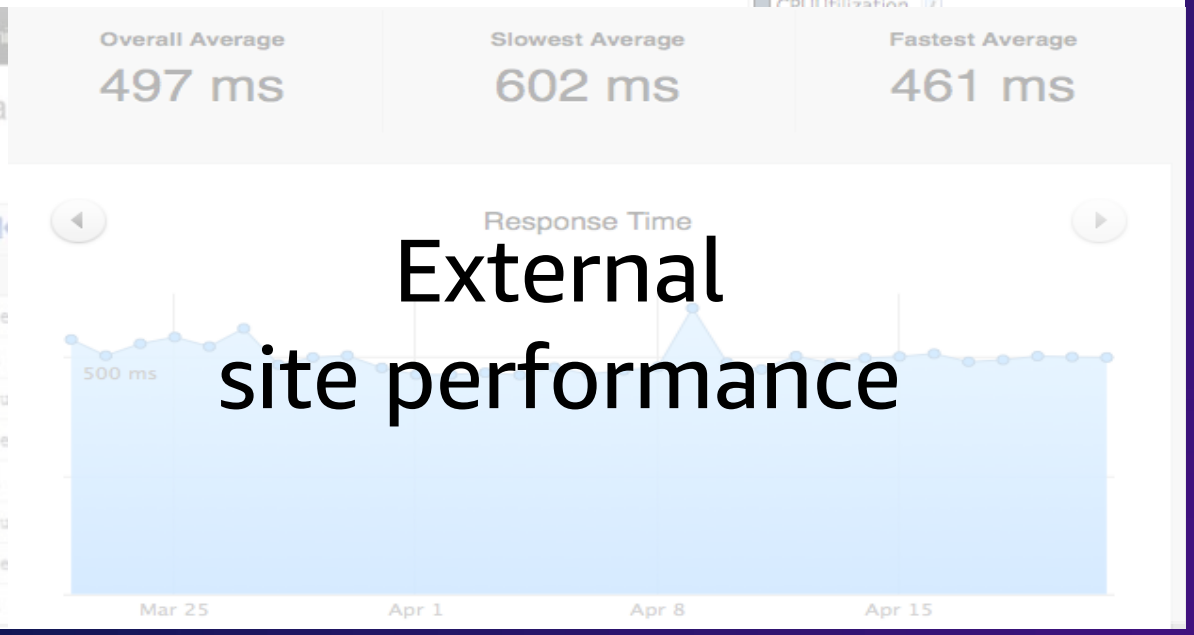

# Host-level metrics




# Aggregate-level metrics




# Log analysis




# External site performance

# Amazon CloudWatch

Amazon
CloudWatch

- Collect: Metrics and logs
- Monitor: Alarms and dashboards
- Act: Auto scaling and events
- Analyze: Trends and metric math
- Compliance and security

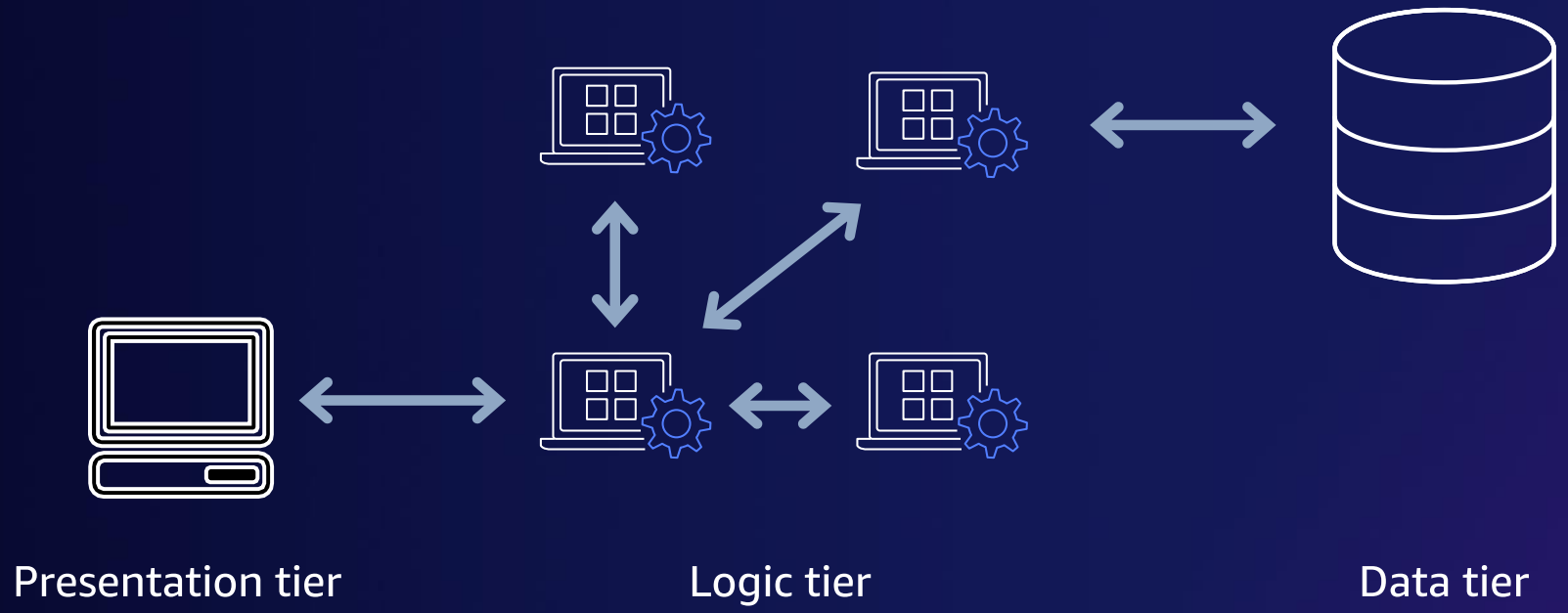
# There are further improvements to be made in breaking apart our web/app layer

# Monolithic architecture

- User interface
- Business logic
- Data access

No separation

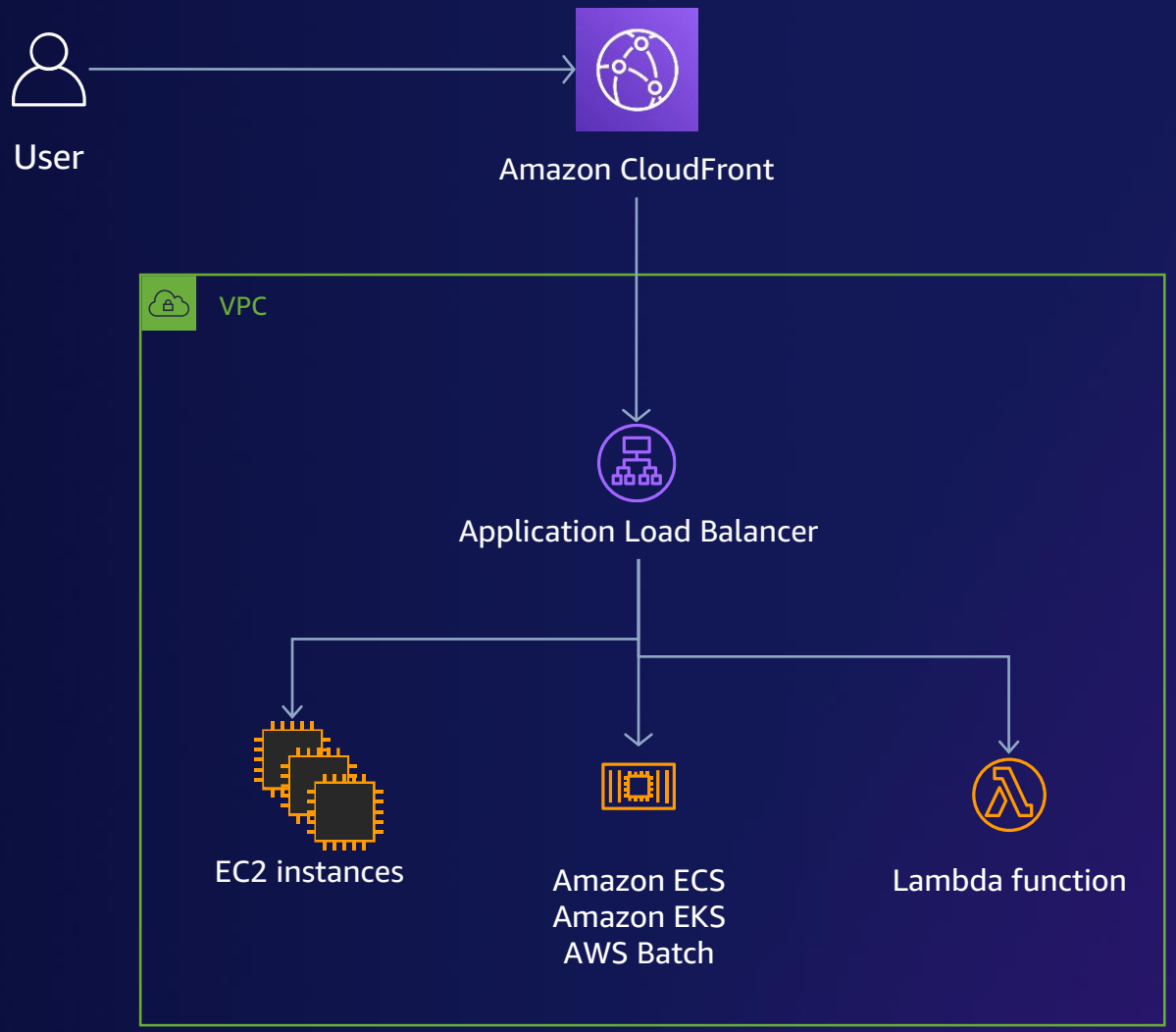
# Service-oriented architecture




## Break monolith into component services

- Treat them separately
- Scale them independently

# Application Modernization with containers / serverless



# Serverless

- Don’t reinvent the wheel

- API
- Queuing
- Transcoding
- Search
- Databases
- Monitoring
- Logging
- Compute
- Machine learning


Amazon API Gateway


Amazon SNS


Amazon Elasticsearch Service


Amazon SQS


AWS Fargate


AWS Lambda


Amazon SES

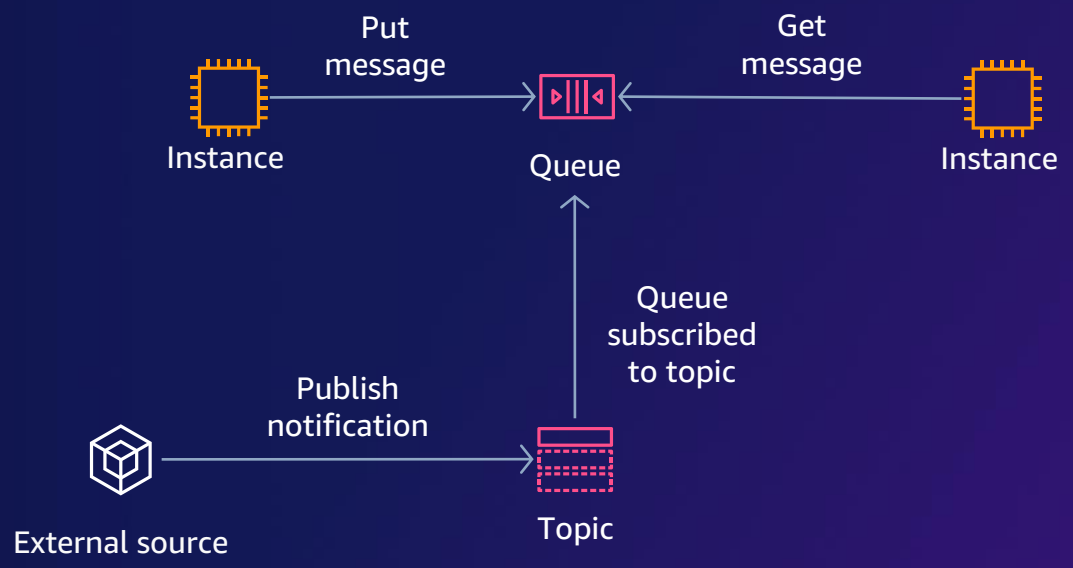

AWS Step Functions


AWS Elemental MediaConvert


Amazon SageMaker

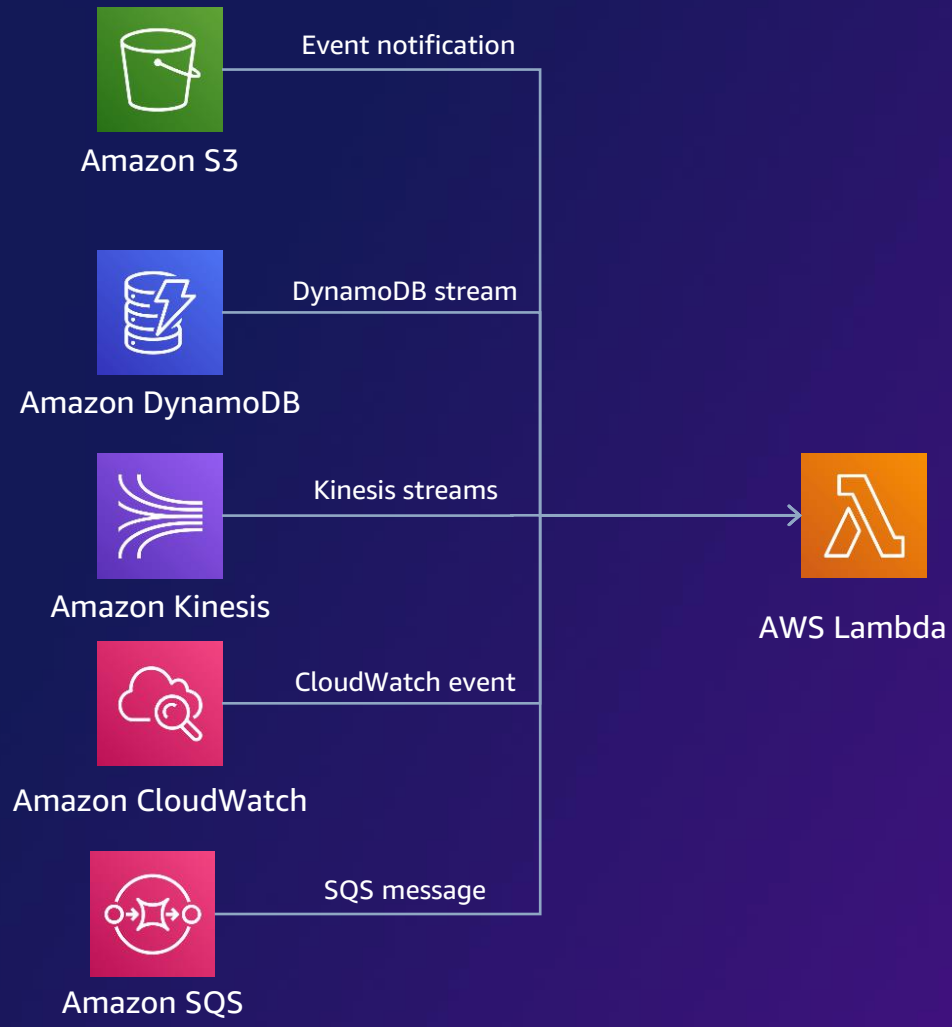
# Loose coupling: Amazon SQS and Amazon SNS

- Reliable (multi-AZ)
- Scalable
- Secure



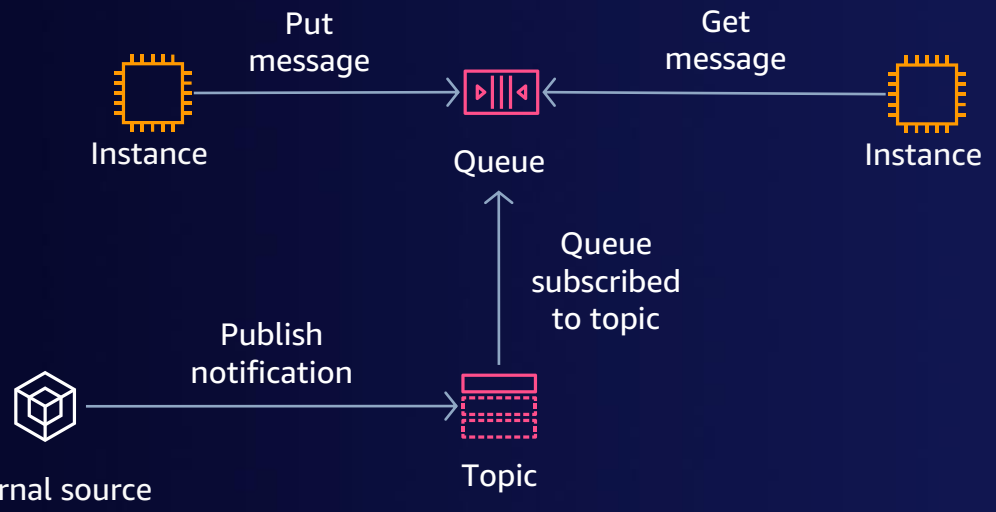
# Event-driven compute: AWS Lambda

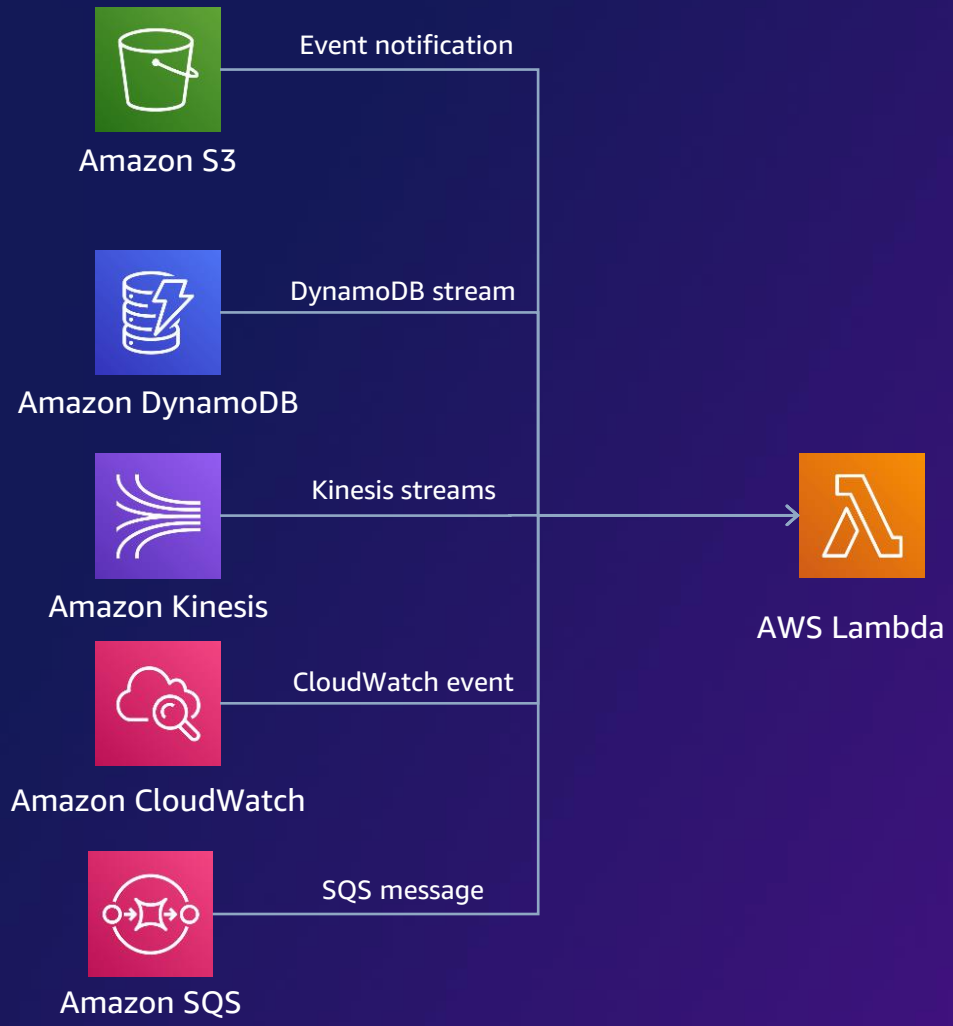
- Functions triggered by events
- Java, Go, PowerShell, Node.js, C#, Ruby, Python
- Serverless
- Implicit scaling



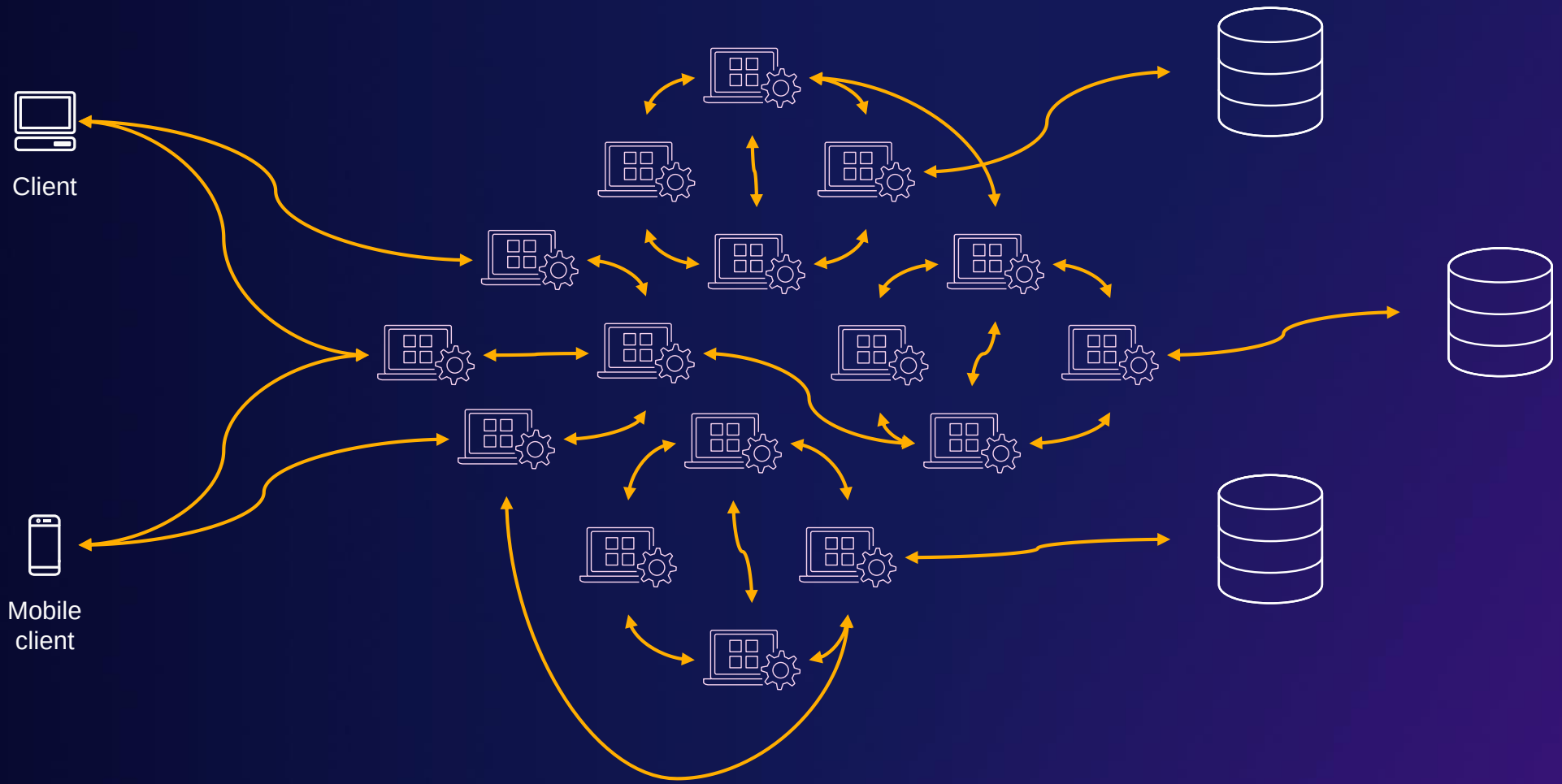
# Loose coupling sets you free

- The looser they're coupled, the bigger they scale
  - Independent components
  - Design everything as a black box
  - Decouple interactions
  - Favor services with built-in redundancy and scalability
    - *Don’t build your own*



# Microservices architecture



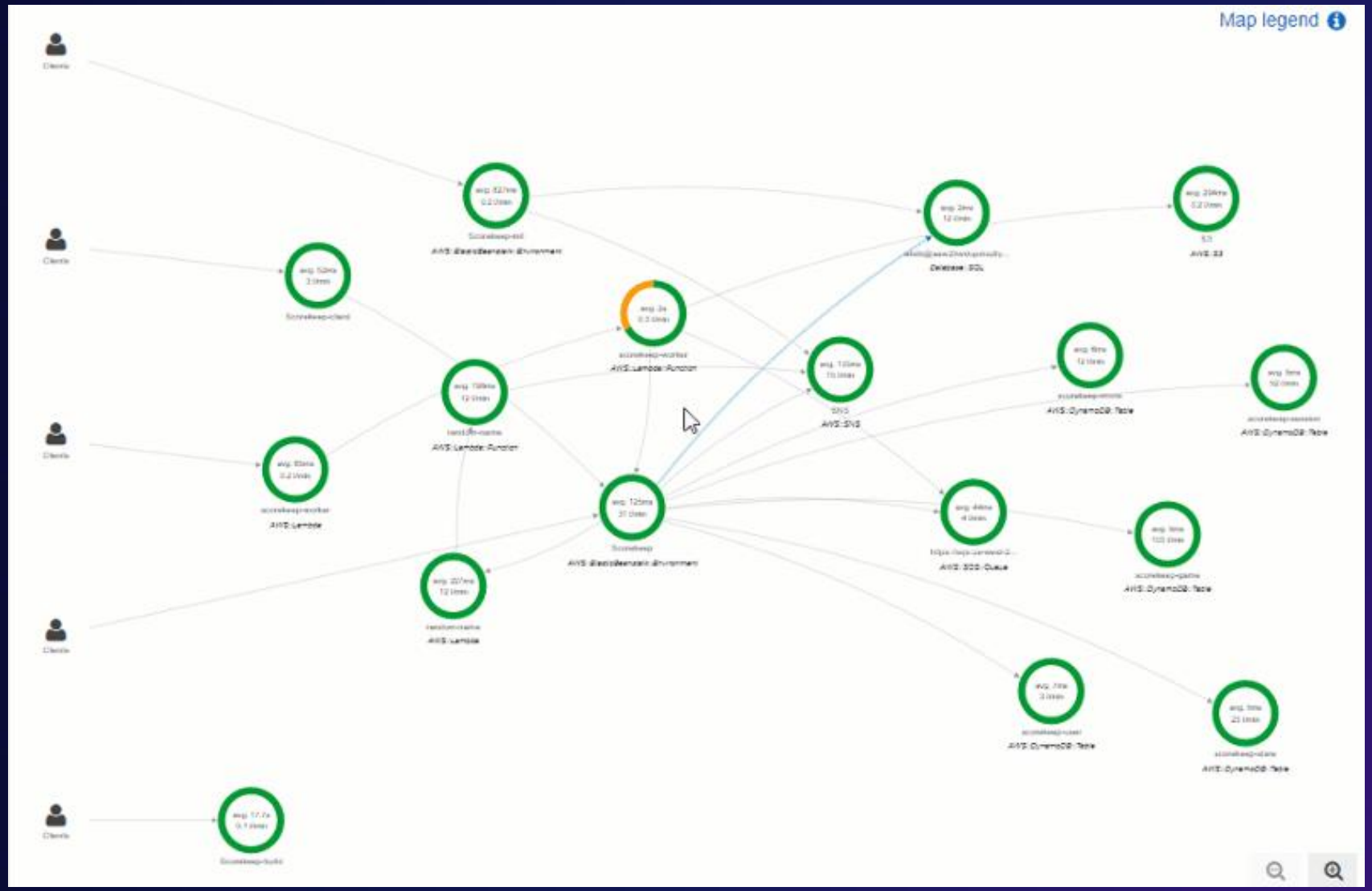
# Microservices architecture on AWS

# AWS X-Ray

AWS X-Ray

- Identify performance bottlenecks and errors
- Pinpoint issues to specific service(s) in your application
- Identify impact of issues on users of the application
- Visualize the service call graph of your application

# Visualize service call graph



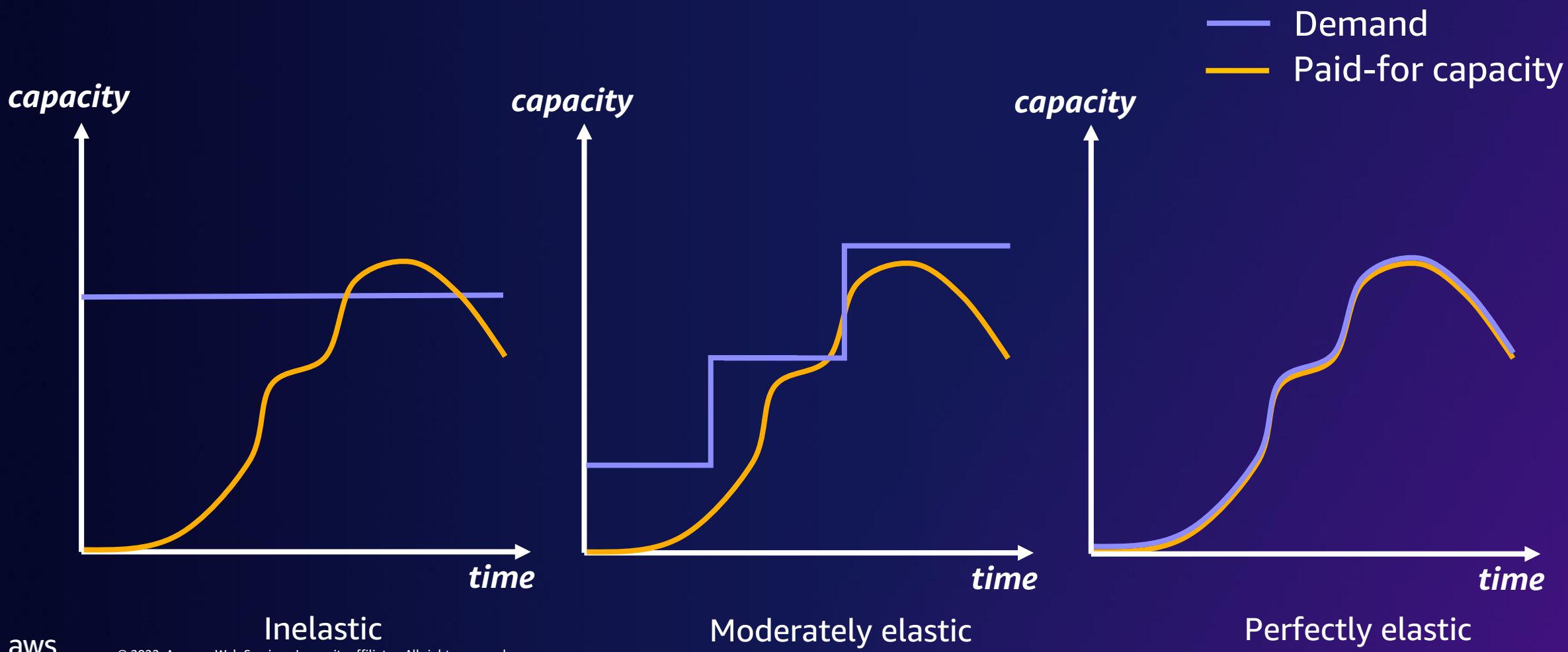
# Elasticity

“The ability to acquire resources as you need them and release resources when you no longer need them.”

-- Source: AWS Well-Architected Framework

# Measuring elasticity

The extent to which paid-for capacity exactly matches demand.




Inelastic

Moderately elastic

Perfectly elastic

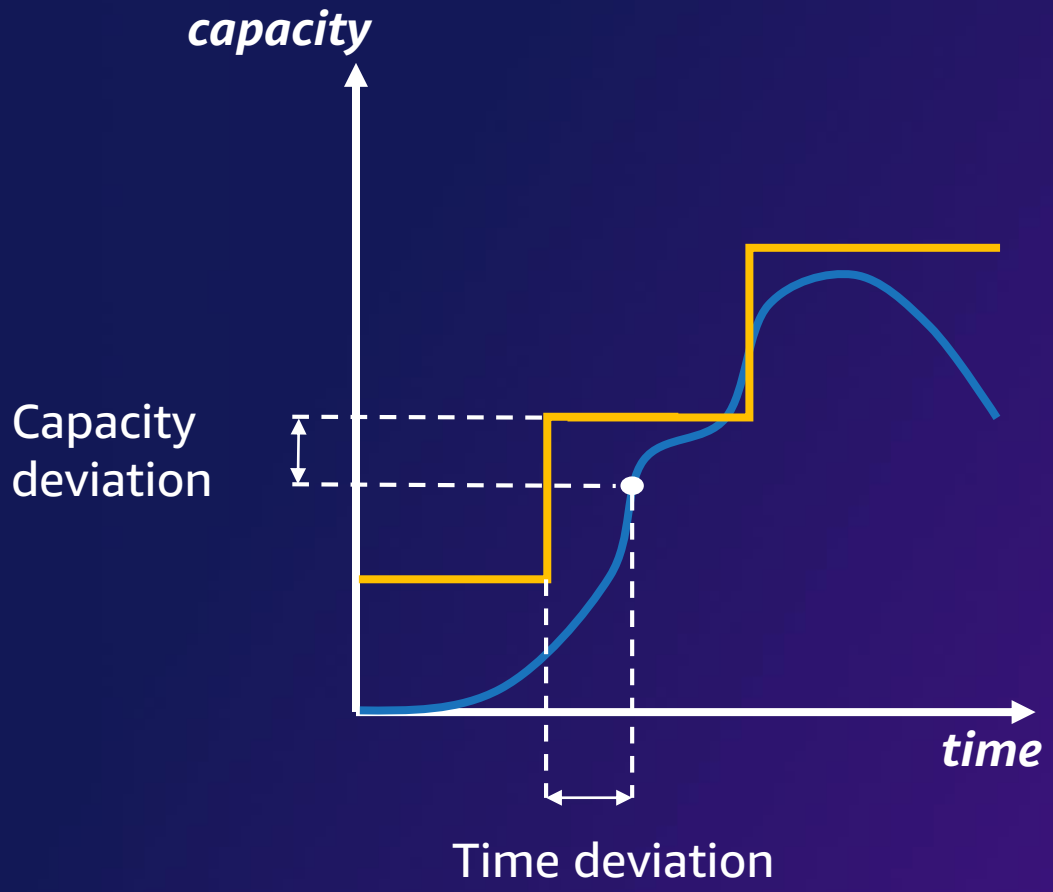
# Factors that determine Elasticity

**Time:**
How quickly can I scale in / out ?

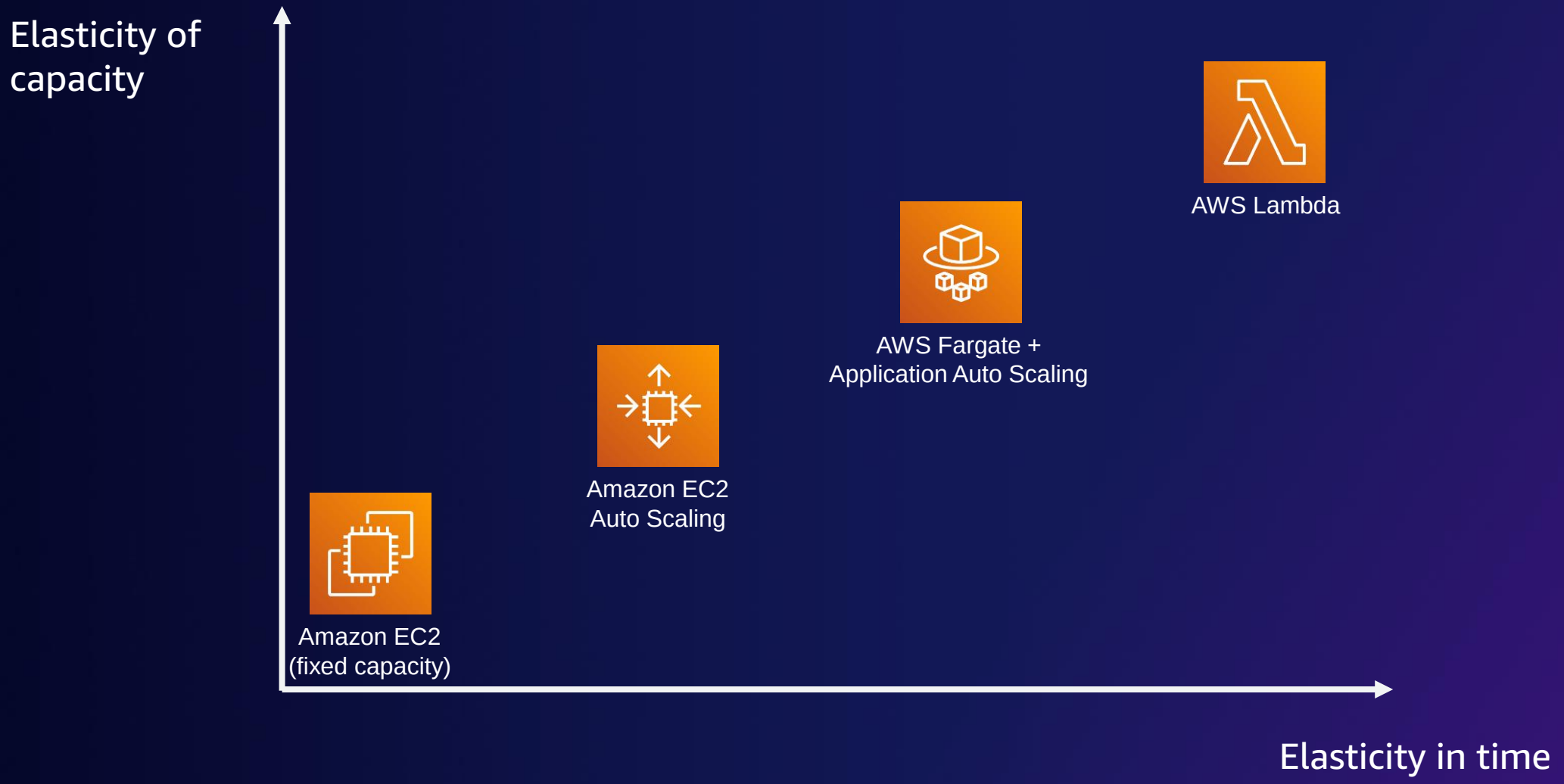
**Capacity Unit:**
By how much can I scale in / out ?

How is **pricing** structured in relation to capacity and time?

- “Billable” capacity is more relevant than “provisioned” capacity



# Elasticity of compute



# Higher Elasticity != Cheaper

## Illustration (fictitious workload)

Amazon EC2

$0.50 per hour
Can process 1M requests/hour at full load

AWS Lambda

Cost $1 per 1M requests

In any given hour:

>500K requests received

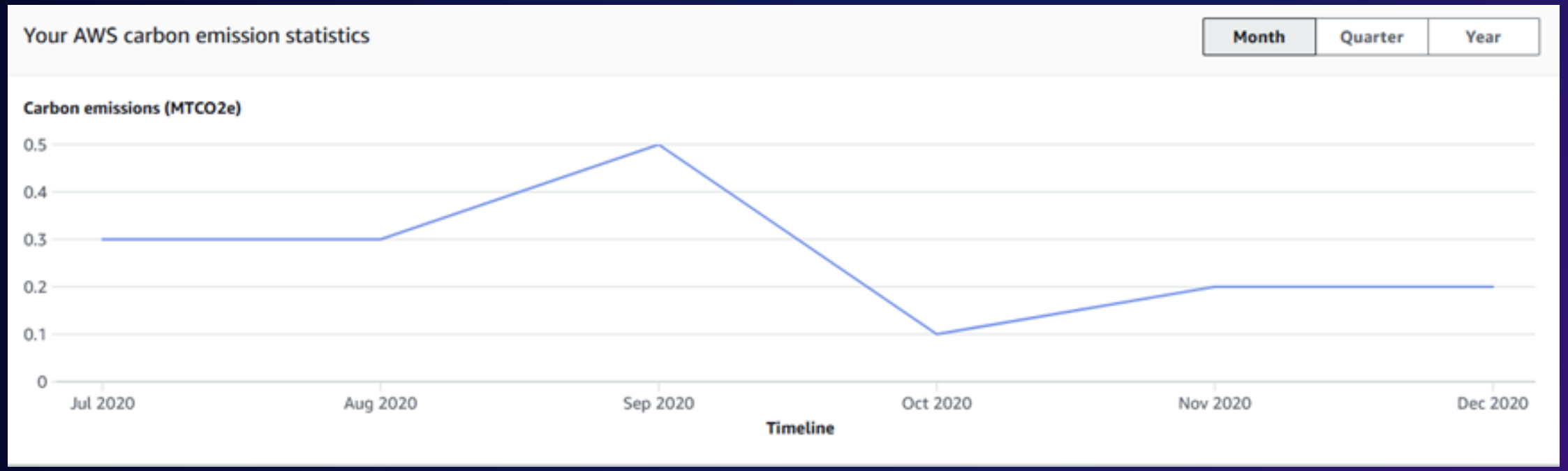
- Amazon EC2 is cheaper

<500K requests received

- AWS Lambda is cheaper

Build a cost model !

# Sustainability




- Carbon emissions are another form of cost.
- Think of it as a different currency: MTCO2e instead of $.
- View your emissions with the AWS Carbon Footprint tool.

# Scaling and Sustainability

c/f AWS Well Architected Framework – Sustainability Pillar

Best Practices:

- Use efficient software designs and architectures to minimize the average resources required per unit of work.
- Implement mechanisms that result in even utilization of components to reduce resources that are idle between tasks and minimize the impact of load spikes.

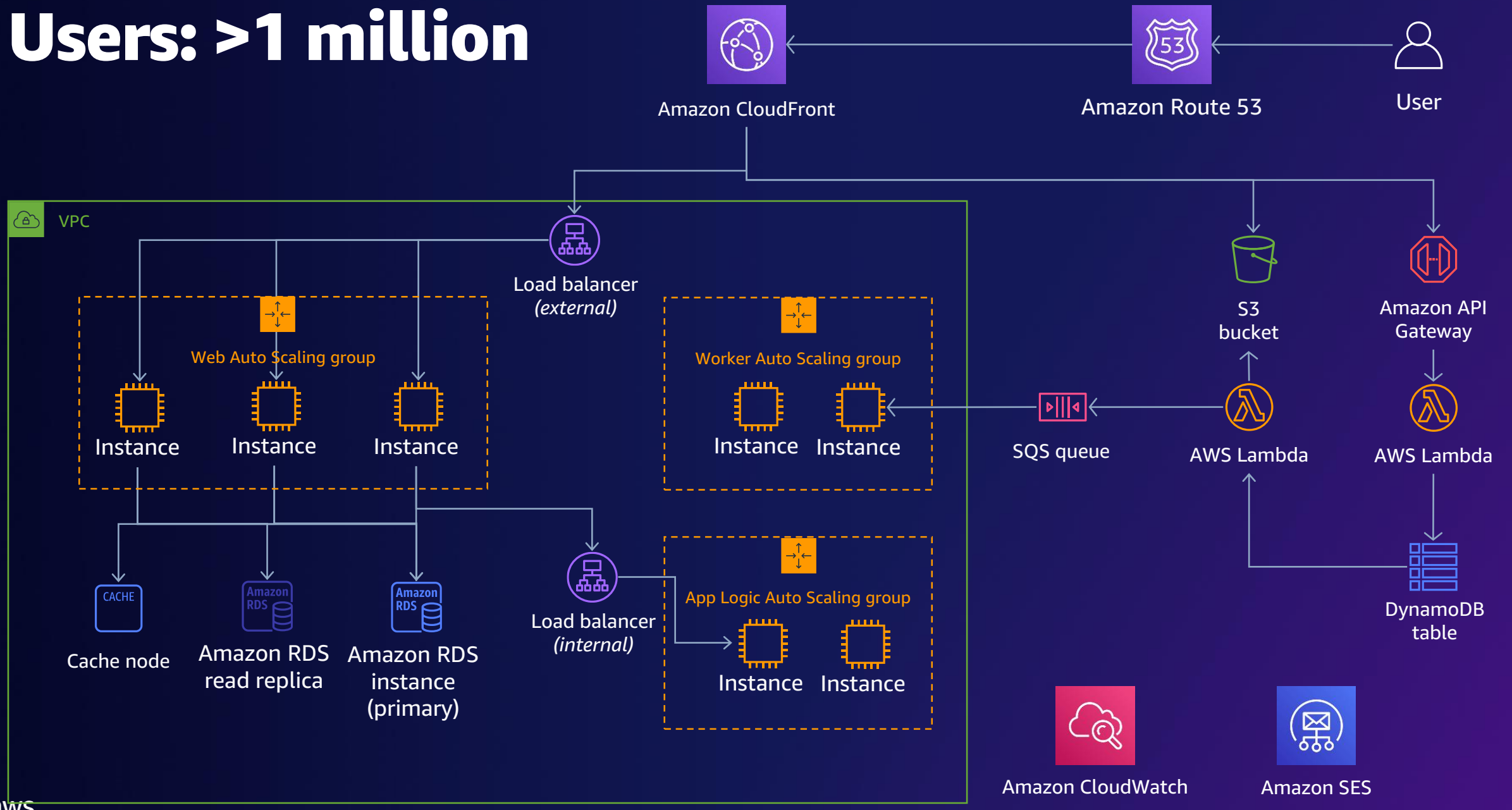
Specific examples:

Increase serialization to flatten utilization across your pipeline. → For inelastic services, use loose coupling to spread load over time and maximize utilization.

Modify the capacity of individual components to prevent idling resources waiting for input. → Wherever possible, use highly elastic services when load is variable, to minimize idle time.

# Beyond 1 million

# Users: >1 million



# > 1 million users

Consider:

- Fine tuning – are you getting the best out of your solution?
- Do you need to go multi-region?
- Is database performance becoming an issue?
- Custom in-house tooling

# Database issues?

Potential solutions

- **Federation:** Split into multiple databases based on function
- **Sharding:** Split one data set across multiple hosts
- Purpose-built DBs: Move some functionality to other types of databases (NoSQL, Graph)
- **Multi-region:** Cross-region replication, possibly with multi-master

# Key messages

- Break your application into component services that can be scaled and managed independently.
- Use managed services to avoid undifferentiated heavy lifting.
- Use edge services to reduce  latency, as well as load on servers.
- Use automation to manage infra, operations, and development.
- Monitor resources and assess efficiency.
- Loose coupling and event-driven compute help you build highly scalable and resilient systems.
- Use serverless to benefit from inherent scalability and availability.
- Build a cost model.
- Factor in your sustainability targets.

# Learn in-demand AWS Cloud skills

## AWS Skill Builder

Access **500+ free** digital courses and Learning Plans

Explore resources with a variety of skill levels and **16+** languages to meet your learning needs

Deepen your skills with digital learning on demand

Train now

## AWS Certifications

Earn an industry-recognized credential

Receive Foundational, Associate, Professional, and Specialty certifications

Join the **AWS Certified community** and get exclusive benefits

Access **new** exam guides

# Thank you!

Dr Mike Rizzo

@rizzomj

https://www.linkedin.com/in/dr-mike-rizzo/

# Please complete the session survey